EKONOMIHÖGSKOLAN
Lunds universitet

Kravhantering inom öppna affärssystem

Kandidatuppsats, 15högskolepoäng, INFK01 i Informatik

Framlagd: *Januari*, 2010

Författare: Alexander Persson
Oscar Sjöcrona

Handledare: Björn Johansson

Examinator: Agneta Olerup
Erik Wallin

Titel: Kravhantering inom öppna affärssystem

Författare: Alexander Persson & Oscar Sjöcrona

Utgivare: Institutionen för Informatik

Handledare: Björn Johansson

Examinatorer: Agneta Olerup

Erik Wallin

Publiceringsår: 2010

Uppsatstyp: Kandidatuppsats

Språk: Svenska

Abstrakt

Det finns ett ökat intresse av att använda Open Source Enterprise Resource Planning (OS-ERP) system inom organisationer idag. En viktig del av utvecklingen av informationssystem är ta fram en kravkontext som tillfredsställer intressenterna i systemet. Kravhanteringen spelar därför en viktig roll för produktresultatet. Kravhantering är dock en mångfacetterad process, och kan bero på många olika faktorer, och det finns ingen enhetlig beskrivande bild inom forskningen av hur kravhanteringsprocessen ser ut, eller hur ska vara strukturerad. Syftet med denna uppsats är att undersöka hur kravhanteringen ser ut inom utvecklingen av OS-ERP. För att göra det tog vi fram en teoretisk grund som bas för en undersökning kring hur kravhanteringsprocessen ser ut. En undersökning genomfördes på fyra organisationer som arbetar med utveckling av OS-ERP. Undersökningen bestod av telefonintervjuer med personer som var insatta i organisationernas kravhantering. Dessa intervjuer analyserades sedan med hjälp av den teoretiska grund som vi hade tagit fram, vilket skedde genom härledning av resultatet utifrån teorin. Utifrån detta kunde vi skapa en övergripande bild av kravhanteringsprocessen som visade på att det finns tendenser till ett likartat tillvägagångssätt för kravhantering inom OS-ERP.

Nyckelord: Kravhantering, ERP, Open Source, krav, systemutveckling

Innehåll

Tabeller

1 Inledning

I takt med IT-utvecklingen har informationssystemen börjat spela en allt större och viktigare roll inom organisationer. Enterprise Resource Planning (ERP) är informationssystem som syftar till att effektivisera en organisation enligt Bjerre (2008). Hon beskriver att konceptet bakom ERP är att dessa system ska innehålla all data och funktionalitet som tidigare har varit spritt på ett antal andra system, för att förbättra verksamhetsprocesserna och öka flexibiliteten inom organisationer. Verville et al. (2007) konstaterar att ett ERP ska stödja informationsflödet genom att ge en holistisk bild över organisationen och dess processer, och fokuserar på processerna istället för funktionerna. Följande definition av ERP ges av Magnusson och Olsson (2008): *”standardiserade verksamhetsövergripande systemstöd" (s.9).* Med detta menar de att ett ERP ska kunna stödja ett brett spektrum av organisationer utan att det ska krävas anpassning av systemet. ERP syftar till att effektivisera affärsprocesserna genom att samla informationen i ett system.

Många organisationer utnyttjar ett ERP system med syfte att få ett övertag över sina konkurrenter (Verville et al. 2007), genom att deras användare får större och snabbare tillgång till den information som finns lagrad. Det finns dock organisationer som är försiktiga eftersom implementering av ett ERP i en organisation tar lång tid och kostar mycket pengar, medan det även finns en stor risk för att det kan misslyckas (Daneva, 2007; The Standish Group, 1995; Verville et al. 2007).

Som alternativ till de traditionella proprietära systemen finns idag system som utvecklats i Open Source (OS). Rapp (2009) pekar på att OS har vuxit sig stort och menar att det fortsätter växa kraftigt eftersom fler och fler organisationer blir intresserade av hur de kan dra nytta av OS i sin organisation. Statskontoret (2003) redogör, i sin rapport om öppen programvara, för en definition över OS.

> *"Med öppen programvara avses programvara där källkoden är fritt tillgänglig och där programmet fritt kan användas, förändras, förbättras, kopieras och distribueras av alla som så önskar."*
> (Statskontoret: 2003, s.9)

Vidare skriver Statskontoret (2003) att det inte är ett krav att OS-baserad mjukvara ska vara gratis, utan betonar främst att inom sådan programvara måste källkoden finnas tillgänglig för den som erhåller produkten. OS regleras i en rad olika licenser som anger de krav och restriktioner kring den kod licensen är kopplad till. Detta betyder att det inte är tillåtet att göra vad som helst med koden även om källkoden är fritt tillgänglig. En annan svensk myndighet, E-delegationen (2009) har tagit fram riktlinjer i sin utredning som bland annat säger att alla svenska myndigheter ska använda sig av öppna standarder i tekniska lösningar. De nämner att i Nederländerna är det krav på både öppna standarder och programvara i offentlig verksamhet. E-delegationen (2009) menar även att öppen programvara alltid ska beaktas som ett alternativ vid val av tekniska lösningar.

1.1 Problemområde

Enligt Fitzgerald (2006) skiljer sig en del av utvecklingen inom OS från vanlig utveckling eftersom de första faserna planering, analys och design vanligtvis inte har några tydliga gränser, och utförs av en utvecklare eller en liten grupp utvecklare. Det kan förklaras av Zhao och Elbaum (2003) som har genomfört en undersökning, där de visar att en stor del av projekten i OS utvecklas av eget intresse och endast en mindre del utvecklas som svar på olika organisationers behov. Lemos (2008) framhäver att OS-ERP är en växande marknad, och att det finns ett ökande kommersiellt intresse för OS. Samtidigt menar Fitzgerald (2006) att på grund av att OS-projekt blir mer kommersiella krävs en högre grad av struktur och kontroll inom utvecklingen, och Sommerville (2007) konstaterar att inom utveckling av större system med hög komplexitet innebär det ett stort problem. Young (2001) ger sin syn på kravhantering inom mjukvaruutveckling:

> *”We often record requirements in a disorganized manner, and we spend far too little time verifying what we do record. We allow change to control us, rather than establishing mechanisms to control change. In short, we fail to establish a solid foundation for the system or software that we intend to build”* (s. xxi)

1.2 Frågeställning och syfte

Vår frågeställning är följande:

Hur fungerar kravhanteringsprocessen vid utvecklandet av Open Source ERP?

Den övergripande bilden över hur processen är strukturerad är en viktig komponent för att besvara vår frågeställning. Förutom detta är det även av intresse att ta reda på hur intressenterna involveras under kravhanteringsprocessen då det i slutändan är de som påverkas av systemet.

Syftet är att presentera ett generellt ramverk för att härleda de olika faserna i kravhanteringsprocessen och sedan applicera detta ramverk på OS-ERP utveckling.

1.3 Avgränsningar

Vi är ute efter att skapa en övergripande bild över ett brett område, vilket innebär att vi endast kommer att skrapa på ytan på de delområden som frågeställningen berör. Vi har således inga intentioner att gå djupt in på hur olika tekniker tillämpas. Dessa kommer dock att beröras för att kunna skapa en bild av kravhanteringen inom utvecklingen av OS-ERP. Vi har valt att se kravhanteringsprocessen som en separat process inom systemutvecklingsprocessen som sker oberoende av dess underliggande struktur. På grund av detta kommer vi icke beröra hur systemutvecklingsprocessen påverkar kravhanteringsprocessen.

2 Teoretisk genomgång

I detta kapitel presenteras den teori som ligger till grund för vår undersökning och analys. Det syftar även till att ge inblick i kravhanteringsprocessen. Kapitlet går inledningsvis in generellt på kravhantering, och därefter kommer det följande avsnittet att behandla de olika faserna som går att urskilja inom kravhanteringen. Sist i kapitlet presenteras ett teoretiskt ramverk baserat på den teori som presenteras tidigare i kapitlet.

2.1 Kravhantering

Hull et al. (2005) ger en bild av kravhanteringsprocessen som en utdragen, kontinuerlig process som följer utvecklingen av ett system. Även Robertson och Robertson (2006) beskriver kravhanteringen som en pågående process, och menar att kraven förändras kontinuerligt, både under systemutvecklingsprocessen och efter det att produkten har släppts.

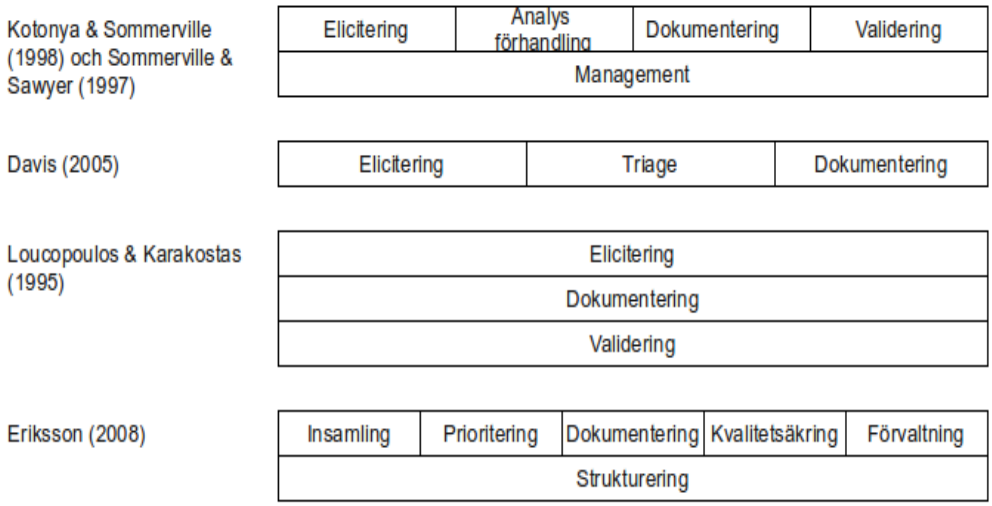
Davis (1993) hänvisar till IEEE Standard 729 som definierar krav som ett nödvändigt förhållande för en användare att lösa ett specifikt problem, och detta förhållande måste uppfyllas av en egenskap i ett system. Robertson och Robertson (2006) definierar krav som: *”A requirement is something the product must do or a quality it must have” (s. 9)* och förklarar att det finns två orsaker till varför krav finns: det första är att produkten kräver det och den andra är att en intressent kräver det.

2.2 Faser

Kravhanteringsprocessen delas in i olika faser, beroende på hur författarna väljer att se på denna process. Kotonya och Sommerville (1998) samt Sommerville och Sawyer (1997) delar in kravhanteringsprocessen i fem olika faser. Författarna menar att de fyra första faserna är efterföljande den föregående medan den femte fasen, management löper parallellt med de tidigare. Vidare nämner de att det inte finns några tydliga gränser mellan de olika faserna och att i verkligheten är de nära knutna till varandra genom att de går in i varandra. Davis (2005) delar in kravhanteringsprocessen i tre olika faser. Han nämner att dessa, speciellt eliciteringen och triage är iterativa aktiviteter som upprepas flera gånger till dess att alla intressenter är nöjda och då det som man kommit överens om dokumenteras. Det Davis (2005) poängterar är att flera managementaktiviteter genomsyrar dessa olika faser. Loucopoulos och Karakostas (1995) delar in

kravhanteringsprocessen i tre processer. De menar att dessa tre processer är knuta till varandra och att de löper samtidigt under hela kravhanteringen. Eriksson (2008) delar in kravhanteringsprocessen i sex faser. Han menar att de olika faserna inte ska ses som en sekventiell process utan ska ses som en iterativ process där varje steg upprepas flera gånger.

I figur 2.1 presenteras en kort sammanfattning över hur några författare väljer att se kravhanteringsprocessen.



Figur 2.1 Bild över hur författarna delar in kravhanteringsprocessen (egen bild)

Även om de olika författarna ger varierande bilder över hur kravhanteringsprocessen fortlöper, syfta de i mångt och mycket på samma aktiviteter som de menar ingår i kravhanteringsprocessen. När Davis (2005) och Loucopoulos och Karakostas (1995) tar upp eliciteringen väljer de att inkludera liknande aktiviteter som Kotonya och Sommerville (1998) samt Sommerville och Sawyer (1997) väljer att presentera i analys- och förhandlingsfasen. Eriksson (2008) väljer dock att dela upp analys- och förhandlingsfasen i två separata faser. Gemensamt med dessa författare är att samtliga menar att gränserna mellan faserna är otydliga, och de pekar på att det finns en nära koppling mellan faserna. Faserna är nära knutna till varandra, och de är även till viss del integrerade i varandra genom att tekniker och aktiviteter överlappar faserna. Därför har även andra författare (Hull et al. 2005; Robertson & Robertson, 2006; Wiegers, 1999) svårt att se klara gränsdragningar mellan de olika faserna.

I denna uppsats kommer vi att utgå från den uppdelning som Kotonya och Sommerville (1998) samt Sommerville och Sawyer (1997) har presenterat. Detta på grund av att deras uppdelning är lätt att följa och vi anser den lämpar sig att använda som upplägg till att beskriva den fortsatta teorin. Dessutom kan det som Davis (2005), Loucopoulos och Karakostas (1995) samt Eriksson (2008) beskriver utan större problem appliceras på denna uppdelning.

2.2.1 Elicitering

Begreppet elicitering åsyftar den process och de aktiviteter som bidrar till att upptäcka och ta fram alla krav som existerar inom en utvecklingskontext (Kotonya & Sommerville, 1998). Loucopoulos och Karakostas (1995) beskriver fasens syfte som att

ta fram all den kunskap som är nödvändig för att skapa en fullständig kravmodell. Davis (2005) skriver att eliciteringen syftar till att skapa förståelse för de problem som användarna upplever. Han menar att eliciteringen ämnar till att bestämma behoven hos intressenterna, och därigenom ta fram alla krav som möjligen går att elicitera.

Baserat på den utvalda litteraturen kring elicitering kan ett antal syften (se tabell 2.1) definieras. Det finns en del skillnader i hur författarna väljer att framhäva olika aktiviteter som karaktäristiska för eliciteringsfasen, dock så finns en överensstämmande bild av vad eliciteringsfasen syftar till att göra. (Kotonya & Sommerville, 1998; Zowghi & Coulin, 2005; Wiegers, 1999). De syften som kan urskiljas kan återfinnas i tabell 2.1.

Tabell 2.1 Syften med eliciteringen

Syften	**Författare**
Identifiera och beskriva organisationsmål, samt prioritering av mål.	Kotonya & Sommerville (1998), Eriksson (2008)
Leta efter källor av krav	Zowghi & Coulin (2005), Eriksson (2008)
Skapa förståelse för omgivningen och användningsområdet.	Kotonya & Sommerville (1998), Zowghi & Coulin (2005), Eriksson (2008)
Identifiera och analysera intressenter, samt prioritering av intressenter.	Kotonya & Sommerville (1998), Zowghi & Coulin (2005), Wiegers (1999)
Välja tekniker baserat på kravkontexten och använda dessa för att ta fram och upptäcka krav	Zowghi & Coulin (2005), Wiegers (1999)
Återanvända krav	Kotonya & Sommerville (1998), Wiegers (1999)
Ta fram krav från intressenter och andra kravkällor	Kotonya & Sommerville (1998), Wiegers (1999), Zowghi & Coulin (2005)

Eliciteringsfasen är dynamisk och därmed uppstår lätt förändringar i kontexten, vilket vanligtvis leder till ny problematik som ska bemötas. Detta ger eliciteringsfasen en iterativ natur eftersom det kan kräva att eliciteringen eventuellt måste återupprepa aktiviteter eller delar av dem, och tillämpa nya tekniker då problemkontexten förändras enligt Hickey och Davis (2003). På grund av den föränderliga kontexten inom kravhantering behöver eliciteringen inte nödvändigtvis följa en specifik metod för kravhantering, utan vikten ligger på att ha en generell förståelse av kravkontexten, specifikt av problemområdet och intressenterna (Kotonya & Sommerville, 1998).

Kotonya och Sommerville (1998) förklarar förhållandet mellan elicitering, analys och förhandling med att de kompletterar och överlappar varandra i ett nära förhållande, som gör gränserna mellan de olika faserna otydliga. Eliciteringsfasen involverar delvis analys av tillämpningsområdet, affärsprocessen och den kontext som produkten kommer att verka inom, medan analysfasen syftar till att hitta problem och konflikter inom de krav som eliciteringen har tagit fram. Sommerville (2007) går ett steg längre, och menar att eliciteringsfasen är integrerad med analys och förhandlings- samt dokumentationsfasen på grund av att deras aktiviteter ligger nära varandra och överlappar genom kravhanteringsprocessens iterativa struktur.

Loucopoulos och Karakostas (1995) lägger tyngd på vad de benämner som eliciteringsfasen och menar att eliciteringsfasen både bör ta fram den kunskap som finns inom kravkontexten, samt även bestämma kravens relevans och skapa djupare förståelse för den betydelse kraven har. Eriksson (2008) menar att det krävs att eliciteringen utförs iterativt eftersom det inte går att samla in alla krav under en kortare period.

Återanvändning av krav

Syftet med återanvändning av krav är att det kan spara resurser eftersom kraven redan är analyserade och validerade (Kotonya & Sommerville, 1998). De konstaterar att analys och validering i mindre omfattning kan säkerställa att kraven går att återanvända inom den nya kontexten. Kotonya och Sommerville (1998) menar även att det är svårare att återanvända krav än vad det är att återanvända design och kod, medan Robertson och Robertson (2006) skriver att inget system är helt unikt vilket gör det möjligt att återanvända krav från tidigare projekt. De menar att även om krav inte går att återanvända i sin helhet, kan de ligga till grund för nya krav.

2.2.2 Analys och förhandling

Analys- och förhandlingsaktiviteter är vanligtvis nära integrerade i varandra (Kotonya & Sommerville, 1998). Vissa författare menar dock att förhandlingsaktiviteter genomsyrar stora delar av kravhanteringsprocessen (Grünbacher & Seyff, 2005; Ahmad, 2008).

Analys
Analysfasens syfte är att hitta problem hos de krav som har tagits fram i eliciteringsfasen. Tillsammans med förhandlingsaktiviteter ska huvudkraven för systemet tas fram för att senare kunna specificeras. (Kotonya & Sommerville, 1998)

Det finns ett antal aktiviteter som anses vara karaktäristiska för analysfasen. Författarna berör ett antal gemensamma aspekter. De aktiviteter som Sommerville och Sawyer (1997), Kotonya och Sommerville (1998), samt Wiegers (1999) väljer att framhäva är:

- Dra gränserna för systemet. Det innebär att både analytiker och andra intressenter måste få fram sina uppfattningar om vart de sätter systemets gränser och sedan genom förhandling komma fram till mer tydliga gränser för systemet.
- Systematisk analys av de insamlade kraven med hjälp av olika tekniker. Leder till att kravens genomförbarhet utreds. Det utreder även huruvida kraven är fullständiga och om de är konsekventa.
- Kravprioritering och klassificering av krav för att ta fram de krav som anses vara viktiga för systemets framgång utifrån den kontext som ligger till grund, samt för att utreda relationer som motverkar eventuella konflikter. Det leder till bättre förståelse för kravkontexten, ökar kravens spårbarhet, samt bestämmer behovet för de enskilda kraven.

Kravprioritering
Prioritering av krav syftar till att säkerställa att utvecklingen av ett system ger högsta värde med lägsta möjliga kostnad. Det innebär att den viktigaste funktionaliteten ska implementeras i ett tidigt skede enligt Wiegers (1999) och Eriksson (2008). I en utveckling som sker stegvis ökar betydelsen av kravprioriteringen enligt Wiegers (1999), och betonar att det vanligtvis handlar om att balansera nytta och kostnad. Kundnöjdhet är en viktig aspekt inom kravprioritering (Karlsson, 1996). Kotonya och Sommerville (1998) väljer att betrakta kravprioritering som en del av både analysen och förhandlingen.

Förhandling
Kotonya och Sommerville (1998) menar att konflikter är en naturlig del av kravhanteringsprocessen, och förhandlingsaktiviteter syftar till att lösa dessa. Resultaten av analysen behandlas i förhandlingen enligt dem. De delar upp förhandlingen i tre processer:

- *Kravdiskussion*: Problematiska krav diskuteras med intressenter.
- *Kravprioritering*: Prioritering av krav sker för att identifiera de viktigaste kraven.
- *Överenskommelse om kraven*: Lösningar till kraven framförs och kompromissande om kraven leder till en överenskommelse.

Förhandling kan betraktas som en faktor som genomsyrar hela kravhanteringsprocessen vars betydelse ökar i korrelation med ökningen av antalet intressenter enligt Ahmad (2008). Kotonya och Sommerville (1998) menar att fasen är integrerad med analysfasen och löper parallellt med denna.

Ahmad (2008) konstaterar att fasen även integreras med prioriteringsaktiviteter och menar att dessa är en fundamental del, vilket även Kotonya & Sommerville (1998) och Wiegers (1999) gör. Davis (2005) benämner förhandlingsfasen och delar av analysfasen som triage, och konstaterar att fasens syfte är att välja ut de krav som ska användas med hänsyn till den kontext som ligger bakom. Han lägger vikt vid kravprioritering och menar att det är essentiellt för fasen.

2.2.3 Dokumentering

Den slutgiltiga produkten efter kravhanteringen är ett dokument där de krav och den funktionalitet som tagits fram kring systemet är bestämda. Detta dokument ligger till grund för all fortsatt planering, design och utveckling av systemet, men utgör även en bas för testning och användardokumentation (Sailor, 1990). Han menar att dokumentation av krav är viktigt för att kravhanteringsprocessen inte ska tappa betydelse.

IEEE Computer Society (1998) har lagt upp rekommendationer för hur kravspecifikationer ska struktureras och skrivas. Deras definition av kravspecifikationen är:

> *”The SRS is a specification for a particular software product, program, or set of programs that performs certain functions in a specific enviroment.”*(IEEE Computer Society: 1998, s.3)

Anledningarna till att en kravspecifikation utformas är många, olika författare betonar olika aspekter. Det finns dock viss konsensus bland dem, och tabell 2.2 visar på vad författarna anser kravspecifikationen syftar till att göra:

Tabell 2.2 Syfte med kravspecifikationen

Syfte	**Författare**
Att kunden kan kontrollera att det är rätt system som utvecklas och vad som kan förväntas.	Sommerville och Sawyer (1997), Wiegers (1999), Davis (2005), Eriksson (2008)
Underlag för projektledarna för att fatta beslut och hantera budgeten.	Sommerville och Sawyer (1997), Wiegers (1999),Davis (2005), Eriksson (2008)
För att utvecklarna ska veta vad de ska utveckla.	Sommerville och Sawyer (1997), Wiegers (1999), Eriksson (2008)
Underlag för testning.	Sommerville och Sawyer (1997), Wiegers (1999), Eriksson (2008)
Används som bas för användarmanualer och manualer.	Wiegers (1999)
Underlag för att utveckla försäljningsstrategier och marknadsföring.	Davis (2005)

Wiegers (1999) menar att hela kravspecifikationen inte behöver vara klar innan utvecklingen av systemet påbörjas. Ibland går det helt enkelt inte att beskriva alla krav från början (Eriksson, 2008). Exempelvis kan en iterativ process fånga upp oklarheter av dessa krav i ett senare skede av utvecklingen och då dokumentera dem. Enligt Sommerville och Sawyer (1997) är det av vikt att en kravspecifikation kan förstås av intressenter och att det finns utrymme för förändringar.

Eriksson (2008) presenterar en lista över olika typer hur kraven kan dokumenteras. Tre av de han nämner, beskrivs i tabell 2.3.

Tabell 2.3 Typer av dokumentation

Dokument	**Beskrivning**
Användarfall	Beskrivning av hur aktörer interagerar med systemet.
Kravspecifikation	Förklarar olika typer av krav till specifik produkt.
Scenario	Består av ett antal användningsfall eller krav.

2.2.4 Validering och verifikation

Boehm (1984) menar att validering syftar till att kontrollera att rätt system konstrueras och verifikation om det konstrueras på rätt sätt. Men enligt Wiegers (1999) finns det en oklarhet kring hur fasen och begreppet validering ska definieras, eftersom validering och verifikation syftar till att utföra samma uppgift.

Valideringen är det slutgiltiga steget i kravhanteringsprocessen. Inom valideringen spelar intressenterna en stor roll. Bland annat genom att de hjälper till med att analysera kravspecifikationen för att upptäcka problem och tvetydigheter i kraven, samt för att utreda huruvida det är rätt krav som har tagits fram (Loucopoulos & Karakostas, 1995; Sommerville & Sawyer 1997).

Wiegers (1999) samt Loucopoulos och Karakostas (1995) menar att valideringen inte är en separat fas som utförs efter de andra, utan en samling aktiviteter som utförs under hela kravhanteringsprocessen. Kotonya och Sommerville (1998) ger en liknande bild, men samtidigt menar de att valideringen och analysfasen har ett antal gemensamma drag. De påpekar dock att det finns skillnader vilket även Sailor (1990) poängterar, där de menar att i analysfasen ligger fokus på att kraven överensstämmer med intressenternas behov och kontrollerar att rätt krav har tagits fram. I valideringen ligger fokus istället på att se till att de krav som tagits fram är rätt beskrivna och att de har uppfattats rätt.

Valideringen är en långdragen process som involverar många intressenter, vilka måste ta ställning till de omfattande dokumenten. Processen kan ta månader att slutföra vid komplexa system och vanligtvis accelereras processen för att utvecklingen ska kunna påbörjas tidigare (Kotonya & Sommerville, 1998). Wiegers (1999) har beskrivit samma problematik med valideringen, och menar att intressenterna vanligtvis inte är intresserade av att lägga tid på valideringen. Eriksson (2008) menar dock att dokumenten kan delas upp till flera olika granskare och på så viss begränsa tiden detta moment tar. Vidare menar han att det är viktigt att upprepa valideringen för att säkerställa att alla oklarheter är eliminerade.

Det är många gånger dyrare att rätta till fel som beror på brister i kravspecifikationen senare i systemutvecklingsprocessen, än under tiden då kraven tas fram (Dorfman, 1990). Innan utvecklingen fortskrider är det viktigt att kraven genomgått en kontroll för att säkerställa att rätt system utvecklas.

De tekniker som går att urskilja ur litteraturen som härleds till valideringsfasen kan ses i tabell 2.4 (s. 13).

2.2.5 Management

Nya krav uppkommer under hela kravhanteringsprocessen (Davis, 2005; Eriksson, 2008; Kotonya & Sommerville, 1998), och managementfasen syftar till att hantera dessa förändringar (Kotonya & Sommerville, 1998). Vilket innebär att managementaktiviteterna löper parallellt för att hantera alla dessa förändringar som uppkommer (Sommerville & Sawyer, 1997).

Management inom kravhanteringsprocessen är främst inriktat mot fyra huvudaktiviteter som är kopplade till den föränderliga kravkontexten enligt Wiegers(1999) samt Sommerville och Sawyer (1997).

- Hantera förändringar i kraven.
- Uppdatera projektets strategi gentemot kraven.
- Hantera versioner av kraven.
- Hantera relationer mellan kraven.

2.3 Intressenter

Robertson och Robertson (2006) och Macaulay (1996) förklarar en intressent vara någon med ett intresse i slutprodukten. Detta innebär att gruppen intressenter kan vara väldigt bred och kan leda till att kravbilden blir mångsidig och komplicerad. Även om alla involverade intressenter inte är relevanta för projektet i samma utsträckning så påverkar alla kravkontexten. Davis (2005) definierar en intressent som en individ eller organisation som påverkas av systemet.

Davis (2005) delar in intressenter i ett flertal kategorier, i rangordning efter viktigast:

- *Kunden*: Den som köper eller erhåller systemet. Kan vara en individ eller organisation.
- *Användaren*: Den som kommer att använda systemet direkt eller indirekt.
- *Marknadsavdelningen*: De som beslutar hur produkten ska positioneras på marknaden.
- *Utvecklare*: De som besitter den mer tekniska kunskapen och vet vad som är möjligt att utveckla.
- *Testare*: De som testar produkten mot de specificerade kraven.
- *Support:* De som har hand om kundsupport och ständigt är i kontakt med användarna.

Decker et al. (2007) menar att intressenter har olika perspektiv, egenskaper, bakgrunder och involveras olika mycket beroende på vilken roll eller vikt de har för projektet, eller för den organisation som slutprodukten ska verka inom. Jiang et al. (2005) konstaterar att valet av tekniker är influerat av intressenters varierade kunskaper.

2.4 Tekniker i kravhanteringen

De tekniker som litteraturen tar upp presenteras i tabell 2.4. Vissa författare delar in teknikerna olika och därför kan samma teknik återfinnas i flera faser. Några tekniker är av den natur att de passar in på aktiviteter som påträffas i flera olika faser. Detta gör att samma teknik kan ha olika syften beroende på vilken fas den utnyttjas inom. Tekniktabellen kommer att delvis användas för att klassificera tekniker utifrån resultatet, och därigenom hitta gemensamma tendenser inom kravhanteringens struktur. Den syftar även delvis att visa på den integration som finns mellan olika faser genom att de tillämpar samma tekniker och aktiviteter för att lösa olika uppgifter.

Tabell 2.4 Tekniker som används i kravhanteringsprocessen

Tekniker	**Beskrivning**	**Författare**	**Fas**
Användarfall	Förklarar olika interaktioner i ett potentiellt system genom scenarios.	Davis (2005), Eriksson (2008)	Elicitering, dokumentering
Checklista	Systematiserar krav och kraven analyseras mot checklistan	Sommerville & Sawyer (1997), Boehm (1984)	Analys och förhandling, validering
Granskning	En grupp, bestående av intressenter och utvecklare, som granskar kraven för att hitta problem.	Sommerville (2007), Bahill och Henderson (2005), Wiegers (1999), Boehm (1984), Eriksson (2008)	Validering
Gruppmöten	Gruppmöten mellan intressenter och utvecklare.	Sillitti & Succi (2005), Macaulay (1996), Robertson & Robertson (2006), Davis (2005), Loucopolous & Karakostas (1995)	Elicitering
Intervjuer	Olika typer av intervjuer med intressenter.	Sillitti & Succi (2005), Kotonya & Sommerville (1998), Robertson & Robertson (2006), Davis (2005), Loucopolous & Karakostas (1995), Boehm (1984), Eriksson (2008)	Elicitering, validering

Fortsättning från föregående sida

Tekniker	**Beskrivning**	**Författare**	**Fas**
Modellering	Med hjälp av modeller och scenario kan man förenkla kraven och delar av systemet och skapa en överblick.	Bahill och Henderson (2005) Boehm (1984)	Validering
Målorientering	Övergripande mål och huvudmål med systemet analyseras och utvecklas.	Sillitti & Succi (2005), Loucopolous & Karakostas (1995)	Elicitering
Personas	Skapar en virtuell karaktär för att upptäcka krav.	Robertson & Robertson (2006), Eriksson (2008)	Elicitering
Prototyper	Intressenter får se en prototyp av produkten och därmed kan de tillhandahålla relevant feedback och detaljerad information.	Sillitti & Succi (2005), Kotonya & Sommerville (1998), Macaulay (1996), Robertson & Robertson (2006), Davis (2005), Boehm (1984), Eriksson (2008)	Elicitering, validering
Scenarios	En berättelse skapas som fungerar som en beskrivning av det nuvarande och det framtida systemet för intressenter.	Sillitti & Succi (2005), Kotonya & Sommerville (1998), Robertson & Robertson (2006), Davis (2005), Boehm (1984), Eriksson (2008)	Elicitering, analys och förhandling, dokumentering, validering
Workshop	Scenarios och öppna diskussioner i nära samarbete med ett antal intressenter för att elicitera krav. Mer formellt och strukturerat än gruppmöten.	Macaulay (1996), Robertson & Robertson (2006), Eriksson (2008)	Elicitering
Ändrings-begäran	Beskriver önskade ändringen för att skapa förståelse för problemet, samt vilka konsekvenser av både beroende på om den införs eller inte.	Eriksson (2008)	Management

2.5 Sammanfattning

Utifrån den teoretiska bakgrunden har vi tagit fram ett teoretiskt ramverk som grund för att analysera resultatet av vår undersökning. Vi kommer därför följaktligen att sammanfatta de teoretiska utgångspunkterna som ramverket är uppbyggt på. Ramverket består av följande faktorer; kravhanteringsprocessen och dess struktur som är baserad på en fasindelning, samt även intressenterna inom den tillfälliga kravkontexten.

Den teoretiska bakgrunden består således av en beskrivning av kravhanteringsprocessen utifrån Kotonya och Sommervilles (1998) uppdelning. Kravhanteringsprocessens struktur inbegriper fem faser, varav fyra som är sekventiella *(elicitering, analys- och förhandling, dokumentering och validering),* och en fas som löper parallellt med samtliga andra faser *(management)*. De nyckelfaktorer som vi hittat var följande:

Tabell 2 Fel! Använd fliken Start om du vill tillämpa 0 för texten som ska visas här..**5 Ramverket**

Faser	**Nyckelfaktorer**
Elicitering	• Identifiering av intressenter och andra kravkällor • Prioritering av intressenter • Kravinhämtning från intressenter och andra kravkällor • Återanvändning av krav • Teknik(-er)/aktiviteter(-er) för att upptäcka krav • Identifiering av fas genom syften med aktiviteter
Analys och förhandling	• Problemanalys av insamlade krav • Kravprioritering • Teknik(-er)/aktivitet(-er) för att analysera krav • Diskussion kring och överrenskommelse om krav • Identifiering av fas genom syften med aktiviteter
Dokumentering	• Val av dokumentationstyp • Identifiering av fas genom syften med aktiviteter
Validering	• Identifiering av fasaktiviteter genom syften med dessa • Teknik(-er)/aktivitet(-er) för att utföra validering
Management	• Hantering av kravförändringar.
Kravhanteringsprocessens struktur	• Fasernas struktur: löpande eller begränsad/koncentrerad under kravhanteringsprocessen. • Fasgränser: Förhållanden mellan faser. • Involvering av intressenter under kravhanteringsprocessens gång.

3 Metod

I detta kapitel redogör vi för den metod vi arbetat efter och diskuterar de val som vi har gjort. Vi presenterar även hur vi hittat de organisationer vi intervjuat och avslutar med en kort diskussion över vårt metodval.

3.1 Val av undersökningsform

Uppsatsens syfte är att beskriva hur kravhanteringsprocessen ser ut inom den OS-baserade ERP-utvecklingen. Kravhantering har studerats i andra sammanhang, men den har inte studerats utifrån den utvalda problemkontexten i någon större omfattning. För att skingra dimman som omger problemområdet krävdes en undersökningsform som gav oss möjlighet att anpassa oss till eventuella skillnader i hur fenomenet såg ut inom den specifika kontexten, i förhållande till hur det beskrivs utifrån teorin.

Eftersom vi är intresserade av att undersöka ett område som är relativt okänt krävs det en ansats som är flexibel och som ger oss möjlighet att gå på djupet där det behövs. Både Jacobsen (2002) och Wärneryd (1993) menar att den kvalitativa ansatsen lämpar sig för osäkra problemområden.

3.2 Val av organisationer

Eftersom undersökningsområdet omgärdades av oklarhet, skapade det osäkerhet kring vem som hade den kunskap vi eftersökte. Därför ansåg vi att det bästa sättet var att göra ett urval av organisationer som arbetar med utveckling av OS-ERP system, och efterfråga personer som hade den kunskap kring kravhanteringsprocessen som vi sökte. Valet av organisationer skedde utifrån ett antal faktorer:

- Mjukvaran ska finnas tillgänglig.
- Mjukvaran ska ha blivit uppdaterad senast i början av juni 2009.
- Projekten ska ha varit listade på Sourceforge.net[1] som aktiva i oktober 2009.

[1] Sourceforge.net är en samlingsplats för open source utvecklare för att underlätta samarbete och sprida open source mjukvara. *http://sourceforge.net/about*

- Möjlighet att genomföra en telefonintervju inom tidsramen för uppsatsen.
- Systemet ska vara anpassat för en internationell marknad.

Vi kontaktade organisationerna i tabell 3.2 och bad om hjälp med att komma i kontakt med rätt personer inom organisationerna som arbetade med utvecklingen av systemet och hade relevant kunskap och erfarenhet kring vårt problemområde. Vi började med att först kontakta dessa organisationer genom deras hemsida, som hade störst antal nedladdningar på Sourceforge.net. Inledningsvis fick vi inte många svar och därför valde vi att gå in på Sourceforge.net igen och därigenom kontakta samtliga personer som var projektadministratörer för de utvalda systemen. Vi fick inte svar från alla men de som svarade hjälpte oss att komma i kontakt med rätt personer, alternativt kunde vissa av dem själva ställa upp och besvara våra frågor. Det fanns dock ett motstånd hos de tillfrågade att göra telefonintervjuer då en stor del föredrog att besvara frågorna via email. Hos vissa saknades även möjlighet att ställa upp på intervjuer inom den tidsram som låg till grund. Därför lyckades vi enbart hitta personer inom två organisationer som var villiga att ställa upp på en telefonintervju. Vi ansåg att antalet var för litet, och därför valde vi att kontakta ytterligare några organisationer som utvecklade system med ett mindre antal nedladdningar på Sourceforge.net, än vad systemen i vårt första urval hade. Intresset visade sig vara något större från dessa, men det var fortfarande enbart ett fåtal som kunde ställa upp på en telefonintervju inom vår tidsram.

I tabell 3.3 har vi en sammanställning över de fyra olika organisationer som slutligen uppfyllde alla våra kriterier. För att hitta dessa fyra har vi kontaktat tretton olika organisationer och skickat ut över 70 förfrågningar via email.

I nästa kapitel går vi närmare in och presenterar de olika organisationerna och hur de arbetar med kravhanteringen i sina projekt.

Tabell 3.3 Organisationerna där intervjuerna genomfördes

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Respondent	Klaus Hofeditz	Marco Schulze	Richard Morley	Si Chen
Position	Medgrundare	VD	Produktansvarig	Projektansvarig
System	Project-Open	Jfire	OpenBravo ERP	Opentaps ERP
Typ av projekt	Kundprojekt	Kund- och interna projekt	Kund- och interna projekt	Kund- och interna projekt
Bilaga	2	3	4	5
Hemsida	http://www.project-open.org/	http://www.nightlabs.com/	http://www.openbravo.com/	http://www.opensourcestrategies.com/

3.3 Typer av intervjuer

Vi hade valt att utföra vad Robson (2002) benämner som semistrukturerade intervjuer för att angripa uppsatsens problemområde. Från början hade vi klart för oss vilka olika teman vi var intresserade av att höra vår respondent berätta om, då vi ansåg dessa som viktiga. En semistrukturerad intervju gav oss möjligheten att ändra inriktning på intervjun beroende på vad respondenten berättar. Dessutom gav våra teman en viss struktur i vårt intervjuande.

Med en helt strukturerad intervju skulle det vara svårt att få fram all den information som vi var ute efter, på grund av att det var väldigt svårt att uppskatta vad som skulle kunna komma fram i intervjun. En hög struktur skulle också kunna hämma vår respondent då vi skulle begränsat respondenten alldeles för mycket med våra frågor. Detta är även något som Wärneryd (1993) tar upp som problematiskt.

Samtidigt skulle en helt öppen intervju möjligtvis inte heller kunna besvara den frågeställning vi har då den öppna intervjun i mångt och mycket är styrd av respondenten.

3.3.1 Telefonintervjuer

Valet av telefonintervjuer baserades på att intervjupersonerna var arbetsamma utanför Sverige och i vissa fall utanför Europa. Därför fanns det vissa begränsningar kring vilka undersökningsmöjligheter som fanns att tillämpa, och därmed ansåg vi att telefonintervjuer var den bästa undersökningsformen med hänsyn till de förhållanden som fanns.

Telefonintervjuer har olika fördelar respektive nackdelar i förhållande till besöksintervjun. Besöksintervjuer anses vara mer tillförlitliga med hänsyn till graden av ärlighet hos intervjuobjektet. En besöksintervju kan eventuellt även underlätta för intervjuobjektet att tala om ämnen som kan anses vara av känslig natur då det skapar en mer förtroendeingivande stämning (Jacobsen, 2002). Telefonintervjuer kan dock motverka det som benämns som intervjuareffekten, det vill säga faktorer som kan påverka respondenten, som uppstår på grund av intervjuarens närvaro.

Eftersom det inte finns mycket forskning inom problemkontexten sedan innan har det varit svårt att skapa sig en förförståelse för vad vi skulle kunna finna. Detta har gjort att vi till stor del har varit tvungna att utgå utifrån en öppen frågeställning under intervjuerna, för att i högre grad kunna anpassa intervjun efter vad vår respondent berättar. Både Jacobsen (2002) och Wärneryd (1993) påpekar dock att telefonintervjuer inte lämpar sig för en öppen intervju utan kräver en viss grad av struktur. Därför har vi lagt upp en struktur med utgångspunkt i de temaområden som vi har tagit fram baserat på den teoretiska grunden, och utifrån dessa ställa kompletterande frågor kring aspekter som utelämnas av respondenten.

3.3.2 Intervjuguide

Vi valde att lägga upp intervjun efter de faser vi gått in på i teorin och benämnde dessa som olika temaområden. Dessa temaområden indelades i ett antal aspekter som vi ansåg karaktärisera faserna utifrån teorin.

Eftersom tanken bakom intervjuerna var att föra en öppen intervju inleddes intervjuguiden med en öppen frågeställning som syftade till att låta respondenten berätta om kravhanteringen inom organisationen när de arbetade med utvecklingen av deras OS-ERP system. Varefter respondenten berättade om kravhanteringsprocessen bockade vi av de aspekter som respondenten berörde. Efter det att respondenten hade beskrivit kravhanteringsprocessen, ställdes frågor baserade på de aspekter inom de temaområden som respondenten inte hade berört. Detta tillvägagångssätt användes för att försöka undvika att styra respondenterna efter vår egen uppfattning av kravhanteringsprocessen.

3.4 Etik

För att hantera de etiska dilemman som Jacobsen (2002) beskriver, har vi gjort följande:

- Innan intervjuerna genomfördes hade vi upprättat en kontakt med respondenterna och klargjort att det var frivilligt att ställa upp. I samband med denna första kontakt sammanfattades även vårt syfte med undersökningen för att respondenterna lättare skulle kunna ta ställning till om de ville vara med.
- Vi erbjöd respondenterna möjligheten till att vara anonyma genom att inte nämna deras namn, organisation eller det system de utvecklade.
- Vi har även försökt presentera våra data på ett riktigt sätt genom att inte ta citat ur sitt sammanhang och genom att delge den data vi samlat in.
- Vi sände även ut transkriberingarna till respondenterna för deras godkännande.

3.5 Analysarbetet

När intervjuerna var genomförda transkriberade vi dem och resultatet framtogs genom analys av empirin. Analysen av det empiriska materialet utfördes genom att varje transkribering lästes igenom ett flertal gånger av oss. Under varje genomläsning markerade vi nyckelord och beskrivningar som vi ansåg var relevanta utifrån det teoretiska ramverket, och därefter diskuterade vi huruvida dessa var relevanta för att beskriva kravhanteringsprocessen. Efter varje genomläsning av varje enskild transkribering sammanställde vi resultaten i ett separat dokument för varje organisation, och dessa reviderades efter varje genomgång. Sedan genomfördes en analys genom att härledning av resultaten utifrån teorin för att jämföra organisationerna utifrån de nyckelfaktorer som ramverket presenterar.

3.6 Diskussion över metodval

Endast fyra intervjuer genomfördes, och vi kunde förmodligen ha fått fler om vi hade haft bredare urvalskriterier från början. Fler intervjuer hade antagligen påverkat resultat och slutsats. Syftet med att begränsa urvalet till att systemen måste ha blivit uppdaterade inom en viss period och vara aktiva, var att hitta organisationer som arbetar med systemen idag. Tanken var att hitta respondenter som aktivt arbetar med utveckling av systemen och därigenom även kravhantering, och därmed förhoppningsvis ha kravhanteringen färskt i minnet.

En del av de organisationer och de personer som vi hänvisades till var ovilliga att genomföra intervjuer via telefon, och ville enbart besvara vår frågeställning via email. Detta ansåg vi vara problematiskt på grund av det öppna förhållningssättet som vi baserade vår undersökning på.

4 Resultat

Följande kapitel ämnar till att kort presentera varje organisation, och följaktligen redovisa hur dessa organisationer går tillväga för att ta fram krav inom deras utveckling av OS-ERP. Fyra organisationer undersöktes genom intervjuer med personer som var insatta i hur kravhanteringen fungerande inom deras egen organisation.

Det finns en distinkt skillnad i hur OS-ERP organisationer tjänar sina pengar gentemot organisationer som erbjuder proprietära ERP. Generellt sett går det att konstatera att den proprietära ERP marknaden tar ut en licenskostnad för varje användare och skapar således ett kassaflöde. I de OS-ERP organisationer vi undersökt finns det inga liknande licenskostnader utan dessa organisationer tjänar snarare pengar på olika kringtjänster som serviceavtal och utveckling av specifika funktioner till kunderna. Skillnaderna i hur organisationer som utvecklar OS och proprietära ERP system skiljer sig kan möjligt påverka hur deras kravhantering ser ut.

4.1 Project-Open

Klaus Hofeditz, medgrundare av Project-Open var vår respondent. Intervjun återfinnes i bilaga 2.

Project-Open utvecklar ett OS-ERP system med samma namn. De erbjuder även olika typer av konsulttjänster kring anpassning av systemet inom organisationer (utöver normal support). Organisationen är baserad i Spanien.

4.1.1 Sammanställning av intervjun

Project-Opens utveckling styrs och baseras vanligtvis på kundens finansiering och krav, och det är kunden som utgör huvudkällan för de krav som utvecklingen bygger på. Vidare tar de även hänsyn till de krav som potentiella kunder har. De har i begränsad skala börjat med egeninitierade projekt utan kund, som är baserad på erfarenheter från tidigare kundprojekt. En del krav uppkommer även internt, det vill säga från dem själva, samt i begränsad omfattning från deras community. Respondenten beskrev prioriteringen av intressenter genom:

> *”Well, you know everybody who a… basically financing and sponsoring that development, will be considered in the first place.”(k6)*

Majoriteten av kraven samlas in på möten med både kunder och de användare som är relaterade till kundprojektet. Under dessa möten anordnar de vanligtvis en workshop hos kundorganisationen för att ta fram krav från kunden och de användare som finns inom den specifika kundorganisationen. Krav inhämtas även delvis från communityforumen på sourceforge.net men eftersom communityn inte står för någon finansiering, är denna kravkälla inte högt prioriterad. De arbetar även med att bygga upp vad respondenten beskriver som ett förråd av krav från fyra andra projekt, och lägga upp dem på hemsidan för att undvika att behöva starta från noll med framtida liknande krav.

I samband med workshopen genomförs en analys som skapar en kostnadsuppskattning kring kraven, varefter de tar fram och använder prototyper för att visualisera kraven som de senare diskuterar med kunden. Det leder vanligtvis till beslut kring vad som ska göras av kraven. Om det uppstår ett behov av kravprioritering på grund av olika begränsningar har kunden alltid sista ordet. Vid situationer där de anser att de behöver mer feedback kring ett specifikt krav inom en kortare tidsperiod, brukar de återupprepa dessa workshopaktiviteter.

> ” *In some very rare cases we would need to organize like a second workshop to dive deeper into particular questions that have come up during this stage and these are only then required when we talk about more complex extensions, configuration of workflows and so on”.(k2)*

Förutom visualisering av kraven genom prototyper, sker det även att de använder dem i samband med någon form av användarfall för att beskriva deras uppfattning av kraven för kunden och därigenom ta fram feedback för att säkerställa att man har förstått kraven rätt. Under kravhanteringsprocessen hanteras de förändringar av krav som uppkommer, genom en iterativ process.

Under de övriga kravhanteringsaktiviteter som utförs sker dokumentation av kraven vanligtvis parallellt genom att specifikationer formuleras i dokument. Dokumentationen används som manual för systemet.

4.2 NightLabs

Marco Schulze, VD på NightLabs var vår respondent. Intervjun återfinnes i bilaga 3.

Nightlabs är en organisation som utvecklar produkter baserade inom OS och erbjuder konsulttjänster inom OS och mjukvaruutveckling. De utvecklar ett OS-ERP system som heter JFire. Nightlabs står själva för support gällande anpassningen av deras egna system. Organisationen är baserad i Tyskland.

4.2.1 Sammanställning av intervjun

Utvecklingen i NightLabs är uppdelad i olika projekt där varje projekt har sin egen kravhantering. Krav tas bland annat fram genom att de pratar med kunden, som är den största kravkällan, och låter kunden berätta om sitt användarfall. Krav inhämtas även

från communityforum där utvecklare och användare kan diskutera och även ge feedback kring presenterade krav. Det uppstår även krav från dem själva, det vill säga från interna källor inom organisationen, och de analyserar även konkurrerande system för att upptäcka krav. NightLabs lägger stor vikt vid att utreda huruvida krav som kommer fram är specifika för en kund eller om det går att anpassa för en större massa.

Återanvändning sker genom att de analyserar krav från projekt kring externa moduler som de har utvecklat tillsammans med en kund. Varefter de försöker utreda huruvida det går att implementera i systemet och göra det användbart för dess användare, med eller utan förändringar.

> *”Of course it happens sometimes, that we think: ‘Ah, this is very specific’ then we implement it specifically and then maybe later someone else comes with more of the same thing, and only some minor changes and then of course we go back and take the previous project, and analyze what can we drag out.”(m24)*

Beroende på vilken utvecklingssituation som råder i organisationen skiljer sig kravhanteringen till viss del. Under enskilda projekt samlas en större mängd krav in i inledningen av ett projekt, därefter erhålls viss feedback som främst gäller förändringar av kravkontext som har framkommit tidigare. Om ett flertal projekt löper parallellt kan det betyda att krav tillkommer löpande på grund av överlappningar mellan projektens krav. Det kommer även in krav kontinuerligt från communityn.

När de har samlat in en mängd krav till ett projekt, försöker de utreda hur kraven kan implementeras och gör en återkoppling till kunderna för att reda ut oklarheter. De menar att kunder vanligtvis inte tillhandahåller tillräcklig information kring sina krav för att det ska gå att implementera direkt.

> *”In this situation we basically have most of the requirements analyses at the beginning and then we have during the project only analyses of maybe misunderstandings and adapting to the requirements, to fit the needs of the costumer better.”(m23)*

De för därför diskussioner med kunder och tar fram användarfall. Dessa användarfall används sedan för att kontrollera att alla krav stämmer överens med vad kunden vill ha och förväntar sig. Denna aktivitet sker regelbundet. Respondenten sade dock att det ofta sker missförstånd och det är viktigt att arbeta nära kunden under hela processen för att snabbt kunna säkerställa att de är på rätt spår.

De presenterar även prototyper bestående av en tidig version av det implementerade kravet för kunden, och erhåller således feedback för att bekräfta om deras uppfattning av kravet är korrekt.

Om begränsningar uppstår kring implementeringen av ett flertal krav inom samma projekt, sker vanligtvis prioriteringsaktiviteter med fokus på vilka krav som ska införas först. De presenterar kravkontexten för kunden, varefter denne får välja ut de delar som ska prioriteras. Andra former av kravprioritering står de själva för och sker internt, och de utgår främst utifrån vad som gynnar projektet baserat på vad som gynnar det stora antalet användare:

> *"So we have to invest from our own money, so we have to decide if this is a benefit for the project in general, for us as a company, whether it benefits a wide range of users or whether if it is a very, very individual thing."(m8)*

De användarfall som har tagits fram fungerar även som kontrakt och dokumentation mellan NightLabs och kunden. Den kommunikation som sker i forumen blir automatiskt sparad och lagrad på en wiki. Kunderna får själv prioritera de krav som de anser är viktigast i deras projekt men de går in och säger vad som är möjligt med tanke på alla de begränsningar som behövs ta hänsyn till:

> *"Of course we make constraints, if he says: 'I want something' and we know that for this something, something else needs to be done and he says: 'Ah, but this I want later' then we have to tell him: 'Yeah, sorry, but you can't build the roof without first building the house'."*

De involverar communityn på liknande sätt genom att hela tiden visa var i processen man är och ta emot feedback från communityn. Förändringar i kraven hanteras bland annat genom vad respondenten benämnde som "change request". Om det uppkommer förändringar av kraven hanteras de olika beroende på vilket skede som de upptäcks i. Är det förändringar av krav som inte blivit implementerade utförs förändringen utan större problem, men om det är en förändring av krav som redan blivit behandlade och implementerade hanterar de detta i nästa version av systemet.

4.3 OpenBravo

Richard Morley, Product Manager på OpenBravo var vår respondent. Intervjun återfinnes i bilaga 4.

OpenBravo är en organisation som utvecklar OS produkter. De utvecklar OS-ERP systemet OpenBravo ERP. Speciell anpassning och annan service kring OpenBravo ERP sker genom partners som är spridda runt om i världen. Organisationen är baserad i Spanien.

4.3.1 Sammanställning av intervjun

Beroende på vilken typ av utveckling det handlar om finns en variation i hur kravinhämtning går till. Kravkällorna består av partners, communityn, samt även interna källor inom OpenBravo. Partners benämns som den främsta källan av krav för utvecklingen av OpenBravo. Kravhanteringen sker genom nära kontakt med beställaren av projektet och är konstant pågående. Samtliga källor spelar dock en roll, respondenten beskriver det som:

> *"Our requirements are driven of what the partners need, but also in a large part by what we believe need to be done strategically to move the product forward and provide a best of breed platform for developers."(r10)*

Om utvecklingen har en partner involverad sker diskussioner med partnern genom möten eller på forum. Andra typer av kravinhämtning sker genom övervakning av communityforum.

Efter att en specifik kravkontext har tagits fram, visualiseras de inhämtade kraven genom olika former av prototyper för att ta fram feedback kring dessa krav, samt för att bekräfta att de framtagna kraven är korrekta. Dessa presenteras sedan antingen för kunder eller för hela communityn, genom möten, forum och bloggar. Prototyperna baseras utifrån de krav som intressenter presenterat. Ibland baseras prototyperna även på en intern kravbild över vad de tror användare vill se i systemet, som de följaktligen presenterar för användare för att ta fram feedback. Respondenten gav följande beskrivning av den processen:

> *”We actually start our requirements gathering from the perspective of what the user will see, what interface, what screen the user will be presented with and the workflows, and we work backwards from there.”(r11)*

Kravprioriteringen sker ibland genom att en partner erbjuder pengar för att specifik funktionalitet ska implementeras, och även genom att krav diskuteras med partnern med syfte att komma till en överenskommelse över vad som ska göras. Det händer även att de utgår utifrån vad de tror vara bäst för användarna, samt att de ibland låter användarna rösta om vilka krav som bör implementeras.

OpenBravo jobbar utifrån vad de kallar sin backlog som är en lista över kraven och dess prioritet. Denna backlog utgörs av tre pelare; första pelaren är ”maintain” som syftar till att ständigt hålla OpenBravo långt fram bland konkurrenterna:

> *”One is that we want to maintain our competitive edge and also that with our partners.”(r29)*

”Delight” som är deras pelare för att sträva efter att underlätta för användarna.

> *”That is we want to develop features and functionality that would delight our users (…)” (r29)*

Sista pelaren är ”monetize” som syftar till att bidra med ett ekonomiskt värde på produkten.

> *”These are features that allow OpenBravo and its partners to make money.”(r29)*

Om en partner är involverad framställs en specifikation över kraven innan utvecklingen startar. Krav som dokumenterats publiceras på ett forum där feedback kan inhämtas och utvecklarna kan ställa frågor till partnern om kravet. Dokumentation av krav sker löpande under projektet, och det sker även parallellt med utvecklingen.

OpenBravo är medvetna om att kraven förändras och har ett flexibelt tillvägagångssätt för att hantera dessa förändringar, fram till dess att utvecklingen av kravet initieras eftersom förändringarna blir mer komplicerade. Förändringar hanteras genom forum, där utvecklare och andra intressenter kan föra en dialog kring förändringar.

> *”And we have a very interactive approach in defining the requirements for our customers, and while that is going on, any change is permissible, until the point that we start development (...)”(r20)*

4.4 Open Source Strategies

Intervju med Si Chen, Projektansvarig på Open Source Strategies. Intervjun återfinnes i bilaga 5.

Open Source Strategies är en organisation som utvecklar Opentaps och vars syfte är att främja OS-utveckling generellt. De tillhandahåller även konsulttjänster, främst anpassning av Opentaps, men även kring evaluering, testning, certifiering, utbildning inom utveckling och integration av OS-mjukvara. Organisationen är baserad i USA.

4.4.1 Sammanställning av intervjun

Huvudkällan för de krav som inhämtas beskrev respondenten som användarna, samt communityn. Andra kravkällor är de interna, det vill säga dem själva under deras egen utveckling och användning av systemet. De har även kundprojekt, där kunden står för kravbilden. Dessa kundprojekt är även de som har högst prioritet.

Kravinsamlingen är delvis koncentrerad genom att de samlar in en större del krav i början genom diskussioner med kunden och dess användare. Samtliga krav under insamlingen lägger de in på vad respondenten benämnde som en issue-tracker. Dock uppkommer nya och förändrade krav under kravhanteringsprocessens gång, främst genom övervakning av communityforumen, och även internt under tiden de utvecklar systemet. Kravinsamlingen sker på olika sätt, bland annat genom att de blir kontaktade av användare som framställer sina behov, som diskuteras för att ta fram en kravbild.

> *“Then basically we would go to the user and discuss exactly what they’re looking for, and we would put it usually on an issue tracker to describe what their requirements are.” (s1)*

Ibland skapas prototyper som presenteras för communityn och baserat på dessa kommer feedback tillbaka från den. Det händer även att de använder scenarios för att beskriva den kravkontext som har uppkommit efter diskussioner med användarna, och använder dessa för att inhämta feedback.

Återanvändning av kraven är också ett inslag i deras kravhantering:

> *”In this way it's a bit like a city: each building in the city represents the requirements of its residents at one point, and the buildings are reused and modified over time to meet new requirements”(s21)*

Förändringar av kraven hanteras genom att införa förändringarna i systemet, som sedan genomgår test och dokumenteras. Detta kan ske på grund av den iterativa process de använder.

> *”It is just an iterative process of the system being used of people and shown to people and their feedback gets incorporated and the system is modified.”(s14)*

De krav som har uppkommit genomgår diskussioner med kunden för att säkerställa att kraven är rätt uppfattade samt genom olika test. Intressenterna på communityforumen ger också feedback genom att olika scenarios presenteras i forumen.

Den kravprioritering som sker, gör det beroende på olika faktorer baserat på om det är ett kundprojekt eller inte. Vid kundprojekt har kunden sista ordet gällande vilka krav som bör prioriteras. Annars styrs prioriteringen av krav genom antalet användare som de tror vill ha funktionen, eller beroende på antalet användare som har kontaktat dem och efterfrågat en specifik funktion.

> *”So for example, a lot of people will come to us and say “ you know we would like this” and that, if it is something that we think will benefit a lot of users we will give it a higher priority.”(s11)*

Dokumentationen sker efter att utvecklingen av mjukvarans funktionalitet är genomförd, och används som användarmanualer samt för att bevara kraven för framtida användning.

> *”Usually after the feature has been implemented, for the purpose of allowing other people to use the feature and also documenting the requirements for later use.”(s20)*

4.5 Inför analys

De resultat som har presenterats ovan är en sammanfattning av de intervjuer vi har genomfört. Resultatframställningen baseras på vår tolkning av intervjuerna och beskriver kravhanteringsprocessen inom varje organisation. I nästkommande kapitel kommer resultaten att analyseras utifrån den teoretiska grunden.

5 Analys och diskussion

Följande kapitel kommer att belysa både likheter och skillnader i vår empiri. Den teoretiska grunden kommer att användas som underlag för att beskriva resultatet, och därmed kunna peka på den underliggande strukturen.

5.1 Elicitering

Det går att urskilja likheter och avvikelser i hur de olika organisationerna samlar in krav. Tre av organisationerna har en mer koncentrerad kravinsamling i början av projektet, men det finns dock en mindre omfattande elicitering som sker löpande i samtliga fall. OpenBravo skiljer sig delvis eftersom de enbart har en löpande eliciteringsprocess där de ständigt upprepar sina aktiviteter för att samla in kraven. Den löpande kravinhämtningen hos samtliga, antyder att de tar hänsyn till en föränderlig kravkontext, i likhet med det Hickey och Davis (2003) samt Kotonya och Sommerville (1998) talar om. Det kan även förklaras av Eriksson (2008) som menar att det är svårt att samla in alla krav samtidigt.

Samtliga organisationer lägger störst vikt på kunderna, vilket även Davis (2005) benämner som en av huvudintressenterna i ett systemutvecklingsprojekt. Det är enbart Open Source Strategies som explicit säger att de involverar användarna när de tar fram krav. Det kan eventuellt förklaras med att användarnas intressen går via kunden och därför involveras inte användarna direkt i de andra organisationerna. Krav tas även fram internt inom organisationerna förutom inom Project-Open. Communityforum används inom samtliga organisationer för att ta fram krav i någon grad, omfattningen av dessas betydelse som kravkälla skiljer sig dock mellan de olika organisationerna. Project-Open involverar inte communityn i kravinsamlingen i någon större utsträckning, medan de andra tar större hänsyn till dem. Det finns ingen klar uppfattning över vilka typer av intressenter som ingår i communityforumen. Men det går dock att spekulera i att det troligtvis handlar om olika grupper av intressenter, bland annat användare. Det kan förklara varför organisationerna inte direkt väljer att gå ut till användarna för att samla in krav. Communityn kan eventuellt även bestå av utvecklare och andra intressenter. Det krävs antagligen ett högt intresse för systemet hos intressenterna, för att de ska vara aktiva på communityn. Därför finns det möjligtvis en obalans mellan olika intressenter på forumen.

Prioriteringen av intressenter är samstämmig inom organisationerna, eftersom samtliga tar störst hänsyn till den som kan tillhandahåller finansiering.

Gruppmöten (se tabell 2.4, sid. 12) är också ett återkommande inslag inom organisationerna, men där Project-Open använder workshops (se tabell 2.4, sid. 12).

Båda aktiviteterna bygger på personliga gruppmöten med diskussioner mellan intressenter och utvecklare. Vidare går det att urskilja diskrepanser i vilka aktiviteter och tekniker som de väljer att tillgå för att elicitera krav. Dessa skillnader kan förklaras med Jiang et al. (2005) som menar att valet av tekniker kan skilja sig på grund av intressenternas kunskaper, eller utifrån andra författare (Hickey & Davis, 2003; Zowghi & Coulin, 2005) som menar att valet av tekniker baseras på kravkontexten.

Återanvändning av krav sker enbart inom tre av organisationerna. Återanvändningen är mest utpräglad inom Project-Open som har valt att bygga ett kravförråd på tidigare projekt, för att undvika att behöva återupprepa processen med att ta fram liknande krav. Vilket ligger i linje med Robertson och Robertson (2006) som menar att återanvändning kan ligga till grund för nya krav. NightLabs återanvänder enbart krav från tidigare moduler, men vanligtvis krävs förändringar i kravet. Deras återanvändning baseras inte på ett kravförråd utan kräver inblick i delar av systemet. Återanvändningen inom Open Source Strategies sker över tiden, utan någon plan. Inom OpenBravo är återanvändningen ingen vanligt förekommande aktivitet, utan återanvändningen av krav sker främst genom återanvändning av kod.

Tabell 5.1 ger en översiktlig bild över hur de olika organisationerna tar fram krav.

Tabell 5.1 Elicitering

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Beskrivning	Varje kravhanteringsprocess inleds med att ett större antal krav samlas in från kunden i början av projektet. Communityn spelar ingen betydande roll för kravinsamlingen.	Kravinsamlingen sker i början av kundprojekten med kunderna och deras användare, men det pågår även en löpande kravinsamling med krav från communityn.	Kravinsamlingen sker löpande i perioder från community, samt från partners. Krav uppkommer även internt i den egna organisationen.	Kravinsamlingen är löpande. Majoriteten av kraven samlas dock in i början av projekten.
Aktiviteter/Tekniker	Workshops, prototyper, communityforum.	Gruppmöten, användarfall, communityforum.	Gruppmöten, prototyper, communityforum.	Gruppmöten, communityforum.
Återanvändning av krav	Bygger upp ett lager av färdiga krav från tidigare projekt. Ska användas för att återanvända färdig-processerade krav, för att undvika att behöver återupprepa process. Kräver vanligtvis förändringar.	Återanvändning krav sker genom analys av tidigare projekt med likande krav. Kräver vanligtivs små förändringar.	Ingen återanvändning av krav.	Viss återanvändning finns då kraven förändras och byter skepnad och de tidigare kraven kan ligga till grund för nya krav.

Fortsättning från föregående sida

Organisation	Project-Open	NightLabs	OpenBravo	Open Source Strategies
Var kommer krav ifrån	Kunder, potentiella kunder, internt, community.	Kunder, internt, community, marknadsanalyser	Partners, community, internt	Kunder, användare, community
Största	Kunderna	Kunderna	Partners, community	Användarna.
Prioritering av	Kunderna	Kunderna	Partners	Kunderna

5.2 Analys och förhandling

Ett återkommande inslag inom analysfasen bland organisationerna är användandet av prototyper (se tabell 2.4, s. 12) för att analysera kraven. De presenteras för olika intressenter för att hitta problem, vilket överensstämmer med det syfte som Kotonya och Sommerville (1998) menar analys- och förhandlingsfasen har. NightLabs avviker dock som inte använder prototyper.

Inom samtliga organisationer finns aktiviteter som syftar till att skapa diskussion kring kraven, vilket är en genomgående faktor inom analys- och förhandlingsfasen. Tre av organisationerna tillämpar gruppmöten, medan Project-Open använder workshops för att föra diskussioner kring kravkontexten. Båda främjar diskussion kring kraven men skiljer sig delvis. Workshop ses som en mer formell och strukturerad aktivitet än gruppmöten, men båda typerna av diskussionsaktiviteter syftar till att komma till en överenskommelse mellan kunden och organisationen om kraven.

Samtliga organisationer använder ett systematiskt tillvägagångssätt för att analysera kraven och hitta problem, med syfte att reda ut om de har tagit fram rätt krav. Det överensstämmer med en av huvudaktiviteterna inom analysfasen som författarna (Sommerville & Sawyer, 1997; Kotonya & Sommerville, 1998; Wiegers, 1999) väljer att framhäva, som menar att det är karaktäristisk del av analys- och förhandlingsfasen.

Det som är gemensamt för organisationerna är att kunden prioriterar kraven inom projekt de är delaktiga i. De organisationer som bedriver interna projekt (se tabell 3.1, s. 16) prioriterar bland annat utifrån vad som gynnar flest användare. Resultatet ligger i linje med det Karlsson (1996) beskriver, som betonar vikten av kundnöjdhet när krav prioriteras. OpenBravo prioriterar främst utifrån vad de benämner som ”monetize”, vilket handlar om att kraven tillåter OpenBravo och dess partners tjäna pengar på systemet.

Organisationernas förhandlingsprocess stämmer överens med den struktur som Kotonya och Sommerville (1998) presenterar där diskussioner, prioritering och överenskommelse ingår.

Tabell 5.2 ger en översiktlig bild över hur analysen och förhandlingen ser ut i de olika organisationerna.

Tabell 5.2 Analys och förhandling

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Beskrivning	Analysen av krav sker genom diskussioner med kunden, som delar av de workshops som utförs i eliciteringsfasen.	Genom att föra diskussioner med intressenter reder de ut oklarheter. Sker efter det att en en mängd krav har tagits fram.	När en kravbild är framställd presenterar de sina prototyper för communityn och partners som ger feedback.	Diskuterar och visar kunden eller användaren funktioner och tar fram feedback. Sker efter att en kravbild är framtagen.
Aktiviteter/Tekniker	Prototyper, workshops	Gruppmöten	Gruppmöten, prototyper, communityforum och bloggar	Gruppmöten, prototyper.
Prioritering av krav	Kundfokuserat där kunden prioriterar kraven i projekten.	Kunden prioriterar sina krav. Nightlabs har ett stort antal parametrar de utgår ifrån när de prioriterar inom sina egna projekt.	Partnern prioriterar när den är involverad och betalar, annars prioriteras krav utifrån de tre pelarna.	Kunderna prioriterar krav i sina projekt. OpenTaps prioriterar efter vad som gynnar flest användare.

5.3 Dokumentering

Dokumenteringen inom Project-Open, NightLabs och OpenBravo sker löpande under kravhanteringen. Den löpande dokumenteringen kan syfta till att vara mer flexibel för att hantera förändringar. Däremot sker dokumenteringen inom Open Source Strategies efter att en funktion har blivit implementerad.

De syften som de olika organisationerna presenterar skiljer sig i stor utsträckning. Inom samtliga organisationer utom OpenBravo ligger dokumenteringen till grund för användarmanualer. Dokumenten används även av NightLabs som kontrakt mellan organisationen och kunden, och inom OpenBravo fungerar dokumentationen som stöd till utvecklarna.

Sättet de olika organisationerna dokumenterar på skiljer sig något åt. Samtliga dokumenterar med kravspecifikationer. Men NightLabs dokumenterar även kraven genom sina användarfall, som Eriksson (2008) klassificerar som dokumentation (se tabell 2.3, s. 9).

Tabell 5.3 ger en överblick över dokumenteringen i de olika organisationerna ser ut.

Tabell 5.3 Dokumentering

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Beskrivning	Dokumenteringen sker löpande och består av specifikationer.	Dokumenterar allt löpande. Dokumenteringen sker genom användarfall.	Dokumenteringen sker löpande med projektet. Kravspecifikationer i projekt kopplas till ett diskussionsforum.	Dokumenteringen sker efter att funktioner blivit implementerade i projekten.
Syfte	Dokumenteringen används som manual för systemet.	Dokumenten fungerar som kontrakt för kunden och hjälp till användaren.	Att ge stöd till utvecklaren för att kommunicera förändringar.	För att bevara kraven och ge hjälp åt användarna.

5.4 Validering

Alla organisationer utom Open Source Strategies använder sig av prototyper för att kunna få feedback på de krav de tagit fram, och av dessa är det enbart Project-Open som inte vänder sig till communityn. Skillnaden mellan organisationerna finns när de presenterar denna prototyp. Tre av organisationerna har valideringsaktiviteter som är återkommande under kravhanteringsprocessen, vilket liknar Wiegers (1999) och Loucopolous och Karakostas (1995) beskrivning av fasen som menar att valideringsaktiviteter utförs under hela kravhantersingprocessen. Dock skiljer sig Open Source Strategies endast utför dessa aktiviteter i slutet.

Samtliga organisationer för en diskussion med kunden genom att utföra granskning (se tabell 2.4, s. 12) av kraven med kunden. Däremot är communityn involverad i valideringen i samtliga organisationer, förutom inom Project-Open som genomgående utesluter communityn från hela kravhanteringsprocessen.

Gemensamt för organisationerna är att de utför de olika aktiviteterna i syfte att kontrollera om de har uppfattat kraven rätt, vilket överensstämmer med det syfte som Kotonya och Sommerville (1998), samt Sailor (1990) beskriver som karaktäristiskt för fasens aktiviteter.

Tabell 5.4 ger en överblick på hur organisationerna hanterar valideringen.

Tabell 5.4 Validering

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Beskrivning	Tar fram feedback och diskuterar kraven med kunden. Sker löpande.	Ta fram feedback och diskuterar den med kunden och communityn. Detta sker regelbundet under projektet.	Eftersom partnern är nära involverad i utvecklingen, och genom frekvent kontakt med communityn, sker det genom frekvent feedback.	Genom diskussion med kunden och communityn. Sker i slutet av kravhanterings-processen.
Aktiviteter/Tekniker	Granskning, prototyper, användarfall.	Communityforum, granskning, prototyper, användarfall.	Communityforum, granskning, prototyper.	Communityforum, granskning, test-case.

5.5 Management

Samtliga organisationer jobbar löpande med att hantera de förändringar som uppkommer under kravhanteringsprocessen, vilket överensstämmer med den bild Kotonya och Sommerville (1998) ger av fasen. Detta antyder att alla organisationer är medvetna om att kravkontexten är föränderlig, något som ett antal författare poängterar (Davis, 2005; Eriksson, 2008; Kotonya & Sommerville, 1998). Alla organisationer tar hänsyn till förändringar om de sker innan utvecklingen har initierats, medan det finns tydliga diskrepanser i hur de arbetar med att hantera förändringarna senare i utvecklingen.

Det går att urskilja två organisationer, NightLabs och OpenBravo, som har tydliga tekniker för att hantera de förändringar som kommer in, och dessa tekniker kan liknas vid ändringsbegäran (se tabell 2.4, s. 12)

Project-Open och Open Source Strategies har inte någon uttalad teknik för att hantera förändringar. De hanterar dock förändringarna som uppkommer genom att båda arbetar utifrån ett iterativt arbetssätt.

Tabell 5.5 ger en överblick över hur organisationerna jobbar med management inom kravhanteringsprocessen.

Tabell 5.5 Management

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Beskrivning	Förändringar tas hänsyn till löpande i kravhanterinsprocessen, enbart mindre förändringar senare.	Förändringar tas hänsyn till om de inte har blivit implementerade, annars behandlas förändringar i nästa version av systemet. Change request används för hantera förändringar.	Förändringar är generellt tillåtna innan utvecklingen startar. Sedan är det upp till kravställaren att ta beslut. Forumen används för hantera förändringarna.	Den iterativa processen som man jobbar efter möjliggör att förändringar i krav kan hanteras. Förändringar genomförs direkt i systemet.

5.6 Fasgränser

Utifrån ett övergripande perspektiv är det svårt att se tydliga gränser mellan faserna. Många av de aktiviteter som organisationerna uppger sig använda återupprepas i skilda faser, och det kan även ske en integration mellan faserna genom att vissa aktiviteter sträcker sig över flera faser. Detta är speciellt tydligt inom OpenBravo som använder prototyper inom faserna elicitering, analys och förhandling, samt validering. Men även Project-Open utmärker sig genom deras användning av samma workshops för elicitering, samt analys och förhandlingsaktiviteter. Inom OpenBravo och Open Source Strategies sker enbart mindre omfattande överlappning mellan dessa två faser. Det är endast inom NightLabs där eliciteringsfasen har mindre koppling till de andra faserna.

I samtliga organisationer överlappar valideringsfasen med analys- och förhandlingsfasen. Denna koppling är något som även Kotonya och Sommerville (1998) beskriver genom att säga att analysen och valideringen har många liknande drag.

I Project-Open och i OpenBravo sker en löpande dokumentering under projektet. Även NightLabs har det, dock med en tydligare koppling till analys- och förhandlingsfasen, samt valideringsfasen. Detta främst eftersom de dokumenterar genom de användarfall som tas fram under dessa faser. Open Source Strategies dokumentering sker främst när en funktion har blivit implementerad.

Det som även går att urskilja är att managementaktiviteter och hantering av förändringar i kraven är något som sker parallellt med samtliga faser i alla organisationer.

Tabell 5.6 ger en överblick på hur organisationerna ser på faserna.

Tabell 5.6 Fasgränser

Organisation	**Project-Open**	**NightLabs**	**OpenBravo**	**Open Source Strategies**
Gränser	Elicitering och analys är parallella. Valideringen överlappar de analytiska aktiviteterna. Dokumenteringen sker parallellt.	Eliciteringen sker vanligtvis i början, sker även i löpande i begränsad omfattning. Analys, dokumentering och validering är nära förankrade i varandra.	Eliciteringen överlappar delvis analysen. Valideringen är integrerad i analysen. Dokumentering sker löpande.	Eliciteringen överlappar till viss del analysen. Analys och valideringen är delvis överlappande. Dokumenteringen sker i slutet av kravhanteringsprocessen.

5.7 Sammanfattning

I detta avsnitt presenteras de gemensamma tendenser vi funnit ur empirin som sedan kommer ligga till grund för slutsatsen.

Elicitering

- En koncentrerad kravinsamling sker i början av projekten.
- Det finns även en löpande kravinsamling.
- De involverade intressenterna är kunder, community, samt interna intressenter. Av dessa är kunderna den prioriterade intressenten.
- Gruppmöten och workshops, samt communityforum är de främsta eliciteringsteknikerna.
- Det finns återanvändning av krav från tidigare projekt.

Analys och förhandling

- Teknikerna gruppmöten och workshops, samt prototyper används för att analysera krav.
- Kunden prioriterar mellan kraven inom kundprojekt. Inom interna projekt prioriteras kraven utifrån vad som gynnar flest användare
- Förhandlingsaktiviteter är genomgående inslag i analysen av krav. Samtliga företag konsulterar intressenter både genom strukturerade diskussionsaktiviteter samt genom mer informella tillvägagångssätt.

Dokumentering

- Sker löpande under stora delar av kravhanteringsprocessen.
- Ligger till grund för användarmanualer.
- Sker genom kravspecifikationer.

Validering

- Sker löpande.
- Granskning och prototyper används för att validera krav.
- Communityn och kunden är involverade.

Management

- Förändringar hanteras löpande fram till dess att utvecklingen initieras.

6 Slutsats

Kapitlet kommer att presentera en bild av kravhanteringsprocessen utifrån den analys och diskussion som gjordes i föregående kapitel. Denna bild kommer visa hur processen fungerar, är strukturerad och hur processen involverar intressenterna. Slutligen förs en diskussion kring slutsatsen, följt av förslag på vidare forskning.

- *Hur fungerar kravhanteringsprocessen vid utvecklandet av Open Source ERP?*

Det finns en distinkt eliciteringsfas som syftar till att upptäcka och ta fram alla krav som finns kring systemet. Eliciteringen sker genom en koncentrerad kravinsamling i början, samt med en löpande kravinsamling som sker i mindre omfattning än den koncentrerade. Det sker även en återanvändning av krav från tidigare projekt. Analys av kraven sker genom analys och förhandlingsfasen. Analysen sker genom nära möten som i samband med att kraven visualiseras för användarna, och dessa ligger till grund för att hitta och bemöta problem. Även valideringen av krav sker genom samma tillvägagångssätt och syftar till att bekräfta om kraven har uppfattats rätt. Prioritering av kraven sker genom att kunderna bestämmer, eller utifrån vad som gynnar flest användare. Under kravhanteringsprocessen finns en dokumentring av kraven, som senare används som underlag för användarna. De kravförändringar som uppkommer hanteras löpande under kravhanteringsprocessen fram till att utvecklingen påbörjas.

Utifrån den empiri vi samlat in går det att urskilja en struktur i kravhanteringsprocessen inom OS-ERP utvecklingen. Det går att konstatera att många av faserna överlappar varandra, vilket gör det svårt att helt bestämma tydliga gränser mellan dem. Analys och förhandlingsfasen överlappar stora delar av eliciteringen och det är en betydande överlappning mellan validering samt analys och förhandlingsfasen. Både dokumenterings- och managementfasen sker parallellt med övriga faser.

Det finns ett antal olika sätt som intressenterna involveras på. De vanligaste teknikerna består av personliga möten med kunder, som syftar till att skapa diskussion kring kraven. Communityforum används för att stimulera dialog kring kravkontexten och ta fram feedback kring systemet. Prototyper används under kravhanteringsprocessen för att visa intressenterna hur arbetet fortgår och för att säkerställa att kraven motsvarar intressenternas behov.

6.1 Diskussion kring slutsatsen

Undersökningen visar på att det finns en strukturerad kravhanteringsprocess inom utvecklingen av OS-ERP. Det framkommer dock tendenser som visar på att kravhanteringsprocessens faser ligger tätt inpå varandra genom att de till stor del löper parallellt (se figur 6.1). Det finns även tendenser som visar på att faserna till stor del är integrerade i varandra, det vill säga att samma aktiviteter används inom olika faser. Det är dock oklart till vilken grad som denna integration sker och utifrån analysen varierar det mellan de olika organisationerna, och därför går det inte att se några gemensamma tendenser för hur nära faserna är integrerade i varandra. Det visar dock på att det finns en nära koppling mellan samtliga faser, och frågan är om vissa faser bör betraktas som separata faser på grund av detta. Till exempel är analys och förhandlingsfasen väl integrerad i valideringsfasen med många likheter och skulle kunna ses och tolkas som en fas istället för två.

Inom den teoretiska bakgrund som vi har använt, har en grundläggande struktur redan existerat. Tanken var dock inte att använda strukturen utan främst uppdelningen av kravhanteringsprocessen. Både strukturen och uppdelningen av kravhanteringsprocessen skiljer sig dock beroende på vilken litteratur som ligger till grund. Möjligen hade det gått att se en annan uppdelning och struktur baserat på de resultat som har fått fram, om vår teori hade utgått från ett annat perspektiv. Slutsatsen visar dock att det finns tendenser till liknande arbetsätt i kravhanteringen inom OS-ERP utvecklingen, dock främst utifrån en mer övergripande nivå.

6.2 Vidare forskning

Det vi kommit fram till är baserat på ett begränsat antal respondenter. Dessa är inte tillräckliga för att dra generella slutsatser om en större population men kan däremot visa på tendenser på hur det kan se ut. Ett förslag till vidare forskning är att undersöka om dessa tendenser vi funnit verkligen stämmer in på en större population. Ett annat förslag är att göra liknande undersökning på proprietära ERP system för att se om det finns likheter och skillnader i hur kravhanteringsprocessen ser ut.

Bilaga 1. Intervjuguide

Hello -------, thank you for participating in our interview and survey.

1. Can we use your name and your companies name in our thesis?
2. Is it okey if we record the interview?

(*Mer öppet tillvägagångssätt, låt respondenten berätta utifrån egna referensramar*):

Frågor om kravhanteringsprocessen

- Can you shortly tell us how your company works with requirements engineering,
- How do you elicit and discover requirements?
    - What are the major activitities or techniques in this phase? *(Ställ frågan om intervjuobjektet själv inte går in på det)*
- What is the next step? *(fortsätt tills hela kravhanteringsprocessen är beskriven utifrån hans ramar)*

Tema-relaterade områden att ställa frågor utifrån. Fråga om respondenten utelämnar någon av dessa aspekter:

Elicitering

- Main source of requirements.
- Other sources.
- Prioritizing stakeholders.
- Involving the stakeholders.
- Reuse of requirements from other projects.

Analys och förhandling

- Finding problems with requirements (ambiguity, incompleteness, conflicts).
- Prioritizing between the requirements.
    - How? Why?
- Deciding when the requirements are complete.

Dokumentering

- Documentation of requirements.
    - Documentation/recording of requirements.
    - Purpose of the documentation.
    - Validate.
        - Making sure the specified requirements are the ones the stakeholders wants.

Management

- Responding to change in the requirements
- Insurance that the development is based on the latest requirements.

Bilaga 2. Intervju med Klaus Hofeditz, Project-Open

O1: Yeah, how do you elicit and discover requirements?

K1: Okay, here's the thing, I think it works very similar in most other open source organizations. I would be surprised if it would be very, very different from what we experienced. Well, it might be. Because [oklart] main differences from Project-Open and a tool like OpenBravo for example is that the entries of barrier, I would consider a little bit lower in our case, we, in particular since we offering again for the second time in our history, a windows installer so that makes it pretty easy to install Project-Open in an open environment, you know on a laptop, there is not much requirements in terms of memory and disc space there. I think we also being an ERP system, we have relatively steep learning curve here with our costumers. There are a couple of issues, people are stumbling over. We try to address them, you know whenever we get the chance to do so. We trying to eliminate them, but you know once the application is installed in an open environment, people start exploring themselves, depending on the user type who is doing this exploration, it takes sometimes, sometimes it works faster, sometimes it takes a bit longer, we offering context help with the most recent versions so these are direct links into an online documentation wiki, which further facilitates handling the application and finding the way around in our application. So that means that there is already before somebody is contacting us, there usually there have been in most of the cases, a very profound evaluation stage, so people are in most cases already aware of functionality they would need to be added, or they would need and they would like to see in the product. And then again depending on the type of users we are talking to, users, some users have less problems describing this functionality to us so that we can come up as a consequence, for example: a html mock up, additional explanations. And in other cases you know we would basically organize workshops, in most cases these are onsite workshops, so we go there for a day, or for one day, the procedure is very, very similar. So we are trying to install as one of the first steps our application on local machines, and we do the evaluation right on that tool, on the way to that we start basically configuring, so we are aiming for some quick gains there. And then after half a day, or in the second day, we go deeper into the process as we are determining key processes that users would like to be supported by Project-Open, and we go into the details, and we basically running a GAP-analysis and bases on the GAP-analysis the outcome of these workshops are usually a cost-estimation about the work that is required to fulfill their requirements. So this is like the big picture, is that something you'd be interested in? Or should we enter more into details?

O2: No, it's fine actually, maybe later we can come with some questions that will focus on some different areas that we have selected from theory then, if that's okay. It was perfect otherwise.

K2: Ok, good great, as I said there is probably not much more to tell. Results of these GAP-analysis are usually these HTML mock ups, that are you know discussed with the client band in an interims phase, before it then basically comes to a decision, that is then made by the client of either a go or a no go. In some very rare cases we would need to organize like a second workshop to dive deeper into particular questions that have come up during this stage and these are only then required when we talk about more complex extensions, configuration of workflows and so on. So this is where we have no test, a kind of a gap also in between expectations clients do have two workflows. In many, many cases they usually know very well what a workflow is, and how it could help them to streamline their business processes. You know having Project-Open providing business support, business process support. But then again what is often in these cases underestimated, is the handling of kind of exceptions and of the rules and handling business processes independent from the system. Or designing the workflow so flexible you know that it is at any given time possible change the basic status of a workflow instance. So this is the experience what we made and yeah that's basically everything I would have to tell you in terms of requirements management or requirements engineering.

O3: Okay, then we have some specific questions…what would say is your main source of requirements?

K3: What the main requirements are?

O4: the main source, I mean like…

A1: Where does the requirements come from , is it the user mostly, or the client, costumer?

K4: So yeah, we pretty much have…of course do to do the many compositions we have with clients in the pre-sales stage of projects, we have a pretty good picture of where our spots are, where clients have additional demands. So nowadays when we are going to clients, we are pretty much aware of what they might expect and what they haven't seen in project-open, what simple we do not provide. So there are not many surprises anymore on that and we are trying to formalize the process currently in a way that we are trying to build up the kind of repository of all these requirements and all the analysis we have done in four more projects. Put them up on the Project-Open at our website, so that when the next time a client comes with the same requirement that you know is targeted to something that hadn't been implemented then in the end for whatever reasons, you know that we don't have to start at zero again. So this is something we are currently looking into, but of course the driving forces is always the client and that is also true for any type of development we are doing. We broke this golden rule of open source, last year or this year for the the very first time, where we started to develop modules for a new market, for a new vertical based on what we thought would be very helpful for this organization, so this is true for the new modules in IT-service management of course, since we are providing professional services to our clients as well. We have a pretty good idea on what the requirements on that end are, but a first time in the history of project-open, you know we started developing something based on, that was initiated by us without a client. This not completely true, we made some experiences in a former project, with one of our largest clients we gained a couple of years ago. We developed some functionality already for them, so we gained some experience there, and part of these new development we have undertaken this year have been based on the results of the client project, but it was not the case that we were in continuous contact and we had pretty short feedback cycles from clients, which we usually have. So this was kind of first time that we broke this rule kind of. Otherwise a very common approach is that after we have done this workshop, and the requirements specifications have been documented and agreed on, that we are still doing software development on the client side. So together with the client we are trying to keep that to a limit of maximum two or three weeks where we are really on side, where we are having the subject matter experts sitting next to us, or at least where we have quick access to them, in order then to build this application. So that have been we've made very, very good experience with this approach and whenever it is possible and the client agrees on that we try to also do it in that way.

O5: Okay, can we ask how do you prioritize between different requirements?

K5: You know it's basically since development as I said, is usually client-driven, the client prioritizes the requirements, so the decision, you know if there's limited budget and what requirements would need to be implemented or will be implemented in Project-Open, is then a client decision.

O6: Okay...

A2: Do you have some prioritization about the client, which client will get his functions implemented first.

K6: Well, you know everybody who a… basically financing and sponsoring that development, will be considered in the first place. We haven't in the past any type conflicts where we had to decide you know which client [oklart] first, in the second phase, these type of conflicts we haven't seen in the past. As I said, maybe this is also different in our case, due to the financial structure, and the overall organizational structure we have, since we are an open source project, which is basically privately financed. You know development is nearly always cost-mod-driven, and that is probably different from many of our competitors. You are interviewing, you have interviewed, so I suppose that the minority of them have external sources in order to finance their operations and their development. So from that point of view it might be a little bit different in our particular case.

A3: Actually, we haven't been into how they are funding their development.

K7: I think it's one of the key-questions, and it makes a pretty big difference if I, obviously we talk frequently to the OpenBravo guys, we have relationships to them and, interchanging opinions and experiences with them and I know for example in the case of OpenBravo, since there is external funding, they have a particular budget they can spend on software development and they have to ask these questions. Which is probably what I can imagine as one of the key issues you have your evaluation, your research is based on, to find the right features, find the features needed by most of the user out there, in order to spread the application as fast as possible, as wide as possible. Our approach in that sense is a different.

O7: Okay, are there any other sources of requirements?

K8: Well basically the two main sources are costumers, our potential clients and you know requirements we ourselves have based on the business that we are running, in terms of support. I can't think of any other sources. What would be a candidate, help me out there, maybe I'm missing something.

A4: The wide community of developers maybe, and [oklart]

K9: You know ERP open source models work in that sense a little bit different, as you probably know. Most of the development stuff is done by the core team, and [oklart] organized and either financed by the core team or financed by the costumer. Community typically in open source projects come into the game, when it comes to localization, when it comes to product testing, and these types of questions. We get from time to time, we get some patches we are happy to basically put into our code repository. But in general it can be said that, you know the community participation in the development of like Project-Open, core modules is very, very limited.

O8: Okay, that's interesting…A question about documenting, do you document requirements firstly?

K10: Well I, just by chance, I put, as a reference for you, this is just an example of what I did you know based on a client meeting I had where it wasn't sure if the client basically would contract our services or would be interested in cooperating with us when I gave them kind of the first estimation that might have been involved but purely out of interest, I was taking these requirements, and I tried to basically document them. The result you can see is the link I just pushed you to on Skype, these types of things that is something that we do with clients, just basically pinpoint something of the core issues and shortcomings and then providing them with like html-mock ups and post them to extend the application, that way as it can be seen on these screenshots. I think that serves as pretty good example of how we approach these types of things. Usually I wouldn't put that up on a blog, it would be obviously part of a document we would then deliver to a client, and as I said it wasn't sure to me if the client wanted to enter in like a second stage. In order not to lose this valuable information which I gathered in this meeting, I came up with quick documentation in my blog.

O9: Do you validate these draft documents? Do you understand the question?

K11: Validate, in what sense validate? I think your question maybe refers, let me see if I assume correctly, maybe a question is towards a situation where a client I basically delivering these requirements and we do then validate them, if they had come initially from the client-side. Is that right?

A5: Yes, but also if you discover the requirements for the client, do the client agree that you discovered the right requirements and that you understood them right?

K12: Well, I mean this is like a ping-pong thing obviously, so you saw them get it right in the first place, but visualizing it this way as you see in my blog, basically helps the client much, much better to get an idea of what the functionality would be. Where he can expect additional port lets [oklart], additional links or what so ever, then in the second stage we enter there the usual procedure that we are taking these html mock up, we are pointing down for the first time, you know some kind of user-cases, however you wanna name them and running some kind of desktop test against the mock ups. Do you have a vague idea of how this procedure works or do you want some additional explanations for that?

O10: No, that's good, that's good. Now a question about change management: when requirements change, how do you respond to that?

K13: Yeah of course we always try to achieve a kind of agile development process with the clients, where as mostly clients are looking into fixed-price projects. In very few cases we were able to develop an application on a more agile approach. But as I said, major improvements and extensions, we usually as I described earlier on the client side and with quick access to the experts, so in that sense we, if we can perform these projects that way there is rfc in the after, or short after the initial implementation of the product. RFC obviously are something that come up during the life cycle of the installation, and then it is basically also, they could be treated in the same way as we do the initial development. This is also our preferred way, we having like after three months or after four months, we do a review, we try to go again to a client side, meet up with them, discuss the implementation, the experiences that have been made

during this initial stage. Still on the client side improving things or changing things so that they better fit the requirements that might have changed meanwhile, or that hadn't been captured in the very first place.

O11: Okay, you mentioned you had an agile approach, so you work in iterations I guess?

K14: In a way we do, we try to build packages that are not all too complex and implement them one after another, that is obviously the approach there. But the whole idea of an agile development, is still kind of difficult to communicate to the client, from our experiences once their happy establish a relationship. The chances are higher to any type of further development in a more agile approach, but in the initial stage where we're having a high risk, a relatively high risk, if you compare it to commercial ERP solutions the risk is obviously considerably lower without doubt. But let's say that most of the clients are trying to minimize their risk as much as they can. One of their methods in order to achieve that is then obviously asking for like fixed price projects, and this is the case in nearly all implementations we have done so far.

O12: Okay, we have a question about the requirements process, would you say the requirements process is parallel to your development or is the requirements engineering process more independant? Do you understand my question?

K15: It's basically, it's a very similar question to the agile approach. Of how I understand agile development, is exactly these short feedback cycles and in a way as I explain it we achieve that basically when we do development on the client side. So I think you would regard that as a kind of agile process, but an agile process where specifications had been evaluated, rather profoundly in a stage before that. And there are usually only minor topics that will be then addressed during the development stage. So in opposite to any type of remote work we do, so there feedback are usually not longer than in comparison to development on the client side.

O13: Okay, so to clarify, you kind of work with a bunch of requirements, I mean you take a bunch of requirements than you analyze and then you in parallel to this process?

K16: Right, exactly. The main challenge here is always that in most IT-projects it is the case that you need to, the biggest work is basically to work together with the clients so that the client himself understand what he or what she wants. It is so, in so many cases we experience that where clients come more or less with vague ideas, and you have to help them in the first place to get a picture of what they really want and then the second challenge is obviously to transform that into the product and get a basic idea of how you would implement it architectural wise and user-interface building, and so on. So I still think that these two faces are relatively independent from each other, so there is like a larger phase of specification requirement engineering which results typically HTML-mock ups, and during the development stage only minor changes then are considered.

O14: Okay, that's interesting. We have one question about the discovering of requirements, what would you say are the major activities or techniques for that? You said you were talking to the community, how do you do this for example?

K17: Community doesn't play a big role in requirements engineering, in our particular situation. We have requirement engineering, we perform on particular costumer projects we do and requirements that come out of the community, this is something we manage through sourceforge, so we are using that platform basically to, we are running our own forum on sourceforge, we have our backtracker on sourceforge, we also have our extension requirements there. So this is how we handle the requirements that come out of the community usually, so in a case where a costumer isn't really willing or able to sponsor some type of development. So this is our platform to manage these types of requirements. Maybe some additional words to that, whenever we make a new release, a minor release, we obviously are going through this list at sourceforge, we are checking them. So this is true for the bucklist or for the future requests and whenever it is possible, when we can fit it somehow in, we try to address these requirements obviously. But as said, as a company that relies on revenue streams, continues revenue streams, the usual way requirements find its way into the product is a concrete project with a costumer.

O15: When you work with your clients, how do you do this? Workshops, is this your main technique for eliciting requirements from your clients?

K18: yes, it is usually a workshop setup yes, so depending on the requirements, the participants are changing slightly during the day. But this is actually the way we do it, we sit together and they kind of

explain what they have figured out themselves so far about the application and they wanna know if they really have discovered everything or if there probably is something they haven't seen and they come up with their specifications. We basically write them down, discuss them with them in the first place, and then coming up with a document, which is basically a mixture of a GAP-analysis and a cost-estimation for the project.

Bilaga 3. Intervju med Marco Schulze, NightLabs

M1: How we do analyze the requirements, you mean what process we have to analyze the requirements. Basically we [oklart] our costumers with, actually most of the requirements come from the costumers. At least this is now the situation, now most of the requirements come from the costumers. We talk with them, analyze their business needs and then we decide if the requirements these costumers have are specific yeah, only for this costumer or they are general for whole sector, whole business sector and based on this decision, if it is something very specific or if it is something that could benefit the whole sector. We decide whether to create the functionality in a separate, specific module only for this user, for this costumer or whether we create a new open source module for a whole branch, for a whole business sector.

O1: Okay…

M2: Basically by abstraction, the costumer always tell you his own story. But then you have heard from experience other stories, from other companies and so based on the background you know if this , what they tell you, their current use case, if this is something that is very specific or if it something that could be beneficial to a wider group of users.

O2: Okay, so it is the costumer your main source of requirements? Do you involve the end users in, how do you involve them?

M3: Yes, we do involve the end users. In different ways, I mean we have two tracks where requirements come in from both other developers, and end-users, one is the community side where we have the possibility to take part in discussions in IRC or in forums, you know we have public forums where people can discuss things and tell us about good ideas. Our requirements may have things they want to use JFire for. Then of course custom-work projects, companies who ask us to help them in their situation and who pay us for this. But basically the way we process this two inputs is more or less the same, we always think how we can implement it in a way that benefits as many people as possible. I mean very often you have a very specific requirement for a certain use-case, but with a very small change or a little abstraction. You can make it usable for more people. For example: if you have a customer who wants to track issues [oklart], wants an integration of issue tracking in JFire and with a certain workflow, so this users specific workflow, you can just make it generic module for basically suited for everyone if you workflow configurable for example. Then everyone can use it, not only this specific costumer. Because there is a lot of work involved that is not specific to this costumer.

O3: Okay…

M4: So basically it's a way of abstraction, actually whole JFire is built this way. We have generic modules that provides certain functionality for very, very wide use-case range, for example: user management and organization management, is something that you can use even outside ERP, it has nothing to do with ERP, this is actually where JFire starts and then we have modules taking care of trade-related stuff, purchasing and selling goods and this is still very general. I think basically every company does that, they purchase and sell things. But what exactly they sell, and under which conditions they sell it, this is different. So then we had a kind of multiple layer approach where one module provides abstract classes, interfaces, defining certain contracts, for example: a price configuration in JFire is just the contract of being able to provide the price in a certain situation which means an offer, when the costumer a request for quotation I think it is in English. Then the vendor needs to do this and so there needs to be a price configuration providing this price, but it doesn't say anything as of how this price configuration should get to this price. There the framework, the underlying abstract phase is totally open, and we have the special module, even for the whole sector, even for the whole company, for one company which provides the specific logic of how this price, in this specific situation can be calculated. For example: if it is a fixed price or if it is dependant on a tarrif, if it is cheaper for students, or if it is cheaper if you buy

more entities like more products, Things like this, but these are specific price configurations and the underlying base doesn't really need to know anything about it.

O4: Okay, so after you discovered your requirements, or [oklart], what is the next step. What do you do with the requirements, how do you…do you analyze them or? Do you understand the question?

M5: How we, if the costumer were someone from the community tells us I need this and this, how we deal with it in the next step?

O5: Right

M6: Yeah, the next step is actually that an architect plans how it could be implemented and asks specific questions that are maybe not yet clear, usually when the costumer has an idea of what he needs, he doesn't provide enough information to really plan something through. So you have to get back, there is some discussion necessary and then during this discussion we create a concept of what we will do, with some how to say it…what is [oklart] in English…I'm missing the word. With the use-case specific, With the use-case specific explanation what is the goal, what should be achieved and some technical details of how it can be achieved. So this concept then basically can be taken first by the costumer to check if we understood everything correctly and second if all the open questions are resourced, you can give it to a developer and say: "Ok, I need this". But of course while things are developed, usually we get back to the costumer. Actually we develop agile, like we do not plan everything through to the last detail till the end. But because very often there are misunderstandings, and so while things are implemented you find out that the assumptions you have made about the use-case in the beginning might not be exactly what the user thought when he told you, you know misunderstandings. Very often, that's why we usually work very close with our costumers, presenting the base of development very often. Showing them the features that are already implemented and in order to get feedback if we are on the right track.

O6: Ok, how do you show the costumers? Do you use prototypes or something?

M7: Mostly no prototypes no, mostly we develop the real thing from the beginning. We develop it usually in a way that there's very soon something to show to the costumer and this is then extended with more and more functionality and more and more features. Basically of course first you need the skeleton of what the costumer wants before you can show him something, this is clear. But once the skeleton is done and you have some, you are to manage the entities and maybe already some UI and that brings them together, we show this the costumer to get feedback. If this is really the entities he needs or if he needs more, if the relations are correct or not.

O7: Okay…how do you prioritize between requirements?

M8: Actually this is different, the strategy of prioritization depends on whether it is something from the community or something from the paying costumer. I mean it is pretty clear that it's coming from the community then basically no one pays us. So we have to invest from our own money, so we have to decide if this is a benefit for the project in general, for us as a company, whether it benefits a wide range of users or whether if it is a very, very individual thing. So the prioritization of these things from the community depends on many, many parameters and the requirements from the costumers prioritization is very easy because they pay us, what they want, at what time, and we tell them how much resources we can put into the project and what we can do in what time and then the costumers have to decide what needs to be done first and what needs to be done later, so that is then up to the costumer because he pays us. I think that is very clear.

O8: Okay, yeah we understand. So the costumer he prioritized in the sense that he pays.

M9: Yeah, and the decision is up to the costumer. If he wants ten features till next week and we tell him sorry, we will not be able to do this, in one week we can only do five features, then he has take out the five that he wants till next week and the rest needs to be done later. So then the decision involving prioritizes are up to the costumer. Of course we make constraints, if he says: "I want something" and we know that for this something, something else needs to be done and he says: "Ah, but this I want later" then we have to tell him: "Yeah, sorry, but you can't build the roof without first building the house".

O9: Right, can we ask: do you have other sources of requirements, except for the costumer and the end user?

M10: Besides the costumer and the community?

O10: Yes.

M11: I would say ourselves, basically from two things: one we use our software ourselves, so we find out that certain things would be good for ourselves to put in, and second: market analyses, yeah when we see what competitors are doing, when we see what in general might be required by certain sectors or certain companies. So …how to say: general market analyses, but this is not the primary source of requirements. I would say the primary source of requirements at the moment are really the costumers. But before, this has changed basically, before the primary source was ourselves looking at the market, looking at what makes sense to put into the application so that the application is usefull for many people.

O11: So, about the documentation, do you document your requirements?

M12: Yes, actually we document them in a different ways, first, I already mentioned if it is a customer request, the customer usually have to pay for it obviously.

O12: Right

M13: So in order to prevent misunderstandings, we write sort of concept, telling the customer exactly how we understood what he wants, how we describe how we will do it, maybe with text, maybe with graph or even maybe with some manufactured screenshots how things could look like, and so we the documentation automatically in the process due to the communication with the costumer. Because if you do this in written form, then at the end, you say "the project is done, I have done everything the costumer wanted" and then the customer might say "I know this is missing, this is missing" and you have to say "Sorry we didn't talk about this before" and if you don't have something in written form, like saying in like in an offer, I offer you "I do your requirements in this and that form and it cost this and that money" then of course you have a problem. So we have automatically this documentation. Then from the input of the community, there is usually documentation in the forum in the discussion and they are usually compiled in our wiki. Usually there is for all this discussions, there are issues and there is a issue tracking online on our community website as well as discussions documents, and planning documents and specification documents in the wiki.

O13: Okay, Can we ask…

M14: Sorry to interrupt you again, this is something. As I said, our input of requirements has changed, now that the software is productively used. Before many things as just analyzed by ourselves. We said "Okay" and we just do it. And we had a lot of informal communication inside the company so this is not all documented. But now when we have our primary source of requirements from the community and from the customer, now we have basically everything documented.

O14:Okay, So during you documentation, how do you validate the draft document?

M15: How do you what?

O15: Validate.

M16: validate?

A1: How do you make sure the requirements are correct.

M17: You mean that the requirements of the customer want, or our understanding is correct, or the final implementation meets the requirements?

A2: Your understanding about the requirements.

M18: Actually by closed discussion with the customer. Personally meeting and trying to understand the use-cases and trying to explain for them how we will implement it, and this is actually regularly, even while the project is running, not only at the beginning and then do something and they afterword find out we did had a basic misunderstanding, this is not of course idea but already while the project is running we meeting regularly with the customer.

O16: Okay

M19: And actually, the community works the same way, I mean our software is open source, so if we implement something that has its requirements origin in the community. Then they can always see the current state on how it is implemented and how it looks and feels to use it and they can feedback what they think about it.

O17: Okay, that is interesting. Now we have some management questions. How do you respond to change in the requirements?

M20: Actually, this is handled differently if it is a project in how to say, if it is like, if the change, the required change is only affecting non-produced, non-released code. Or if it is affecting released API, that requires refactoring, then of course, change requests. Change request that are not affecting API, that are not affecting the system data, then it is very easy to just, get it into the normal development flow. But if it require major refactoring then of course then it have to be postponed to the next major release of JFire, this is clear.

O18: Okay, we just have a question regarding, how the process looks, when you, the requirements process, do you discover and elicit requirements all the time, or do you see that kind of, you do it, you do a lot of requirements at one time and then you go to another process, or is it more like one requirement at the time?

M21: You mean when we collect them or when we, how we process them? I didn't really get the question.

O19: My question is kind of..

M22: When we do this, when we look for new requirements, or what do you mean?

O20: Yeah, does it happen frequently during your development or do you do it as one instance and then you go on and analyze and document or do you get the requirements all the time so to speak.

M23: Actually, both, I mean for one thing of course with costumer-work projects, the costumer and tell you what he wants, and then you analyze and you make a certain project plan, what the use-case requires. You analyze this and then you implement it, but of course while implementing you get the feedback from the costumer all the time to detect changes, necessary as early as possible. In this situation we basically have most of the requirements analyses at the beginning and then we have during the project only analyses of maybe misunderstandings and adapting to the requirements, to fit the needs of the costumer better. But on the other hand we multiple projects running like …many costumers might want to use different things in JFire, might want to do different things in JFire, they have some multiple projects running at the same time and they might touch each other sometimes, like maybe one project requires an extension in the basic modules, in one of the basic modules, and maybe another project does [oklart], some coordination required. Which is actually all the time happening, and again from the community of course we have all the time input too, so there's basically always a flowing process.

O21: Okay, we're just gonna ask a question about reuse of requirements from other projects, does it happen?

M24: Yes and no, as I already said when we, when we are told some new requirements. We analyze how we can maybe abstract it to make usable to a wider audience. Of course it happens sometimes, that we think: "Ah, this is very specific" then we implement it specifically and then maybe later someone else comes with more of the same thing, and only some minor changes and then of course we go back and take the previous project, and analyze what can we drag out. One layer down, one abstraction layer down, to cover both projects with the same codes and just, only some minor parts specifically. This is reusage definitely.

O22: okay…

M25: And of course during this process, when the second costumer comes to you wanting something, of course you think about, whether it makes sense to even abstract it more, to have it fit an even wider audience. Because usually if you already start with a new abstraction layer, then very often you can

design it in a way where it does not only cover the two use-cases that you have now concretely to deal with, but even some more.

O23: Okay, just a short question about the agile development: Would you say that the requirements process is parallel to the agile development process or is it…I guess you said before, you said it was connected to the project when you had a deadline.

M26: Actually it is both, I mean the majority of the requirements analysis of course in the beginning of the costum project. But it is in a certain way parallel because very often wild users sees how the project…he thinks about more details, he sees that often there's some more flexibility required, which he didn't think about before. So you always have to adapt the requirements to the current understanding of the projects. Both on the user side and on the developer side, because both develop the understanding while they work on that. So in a certain way it's parallel, but of course a bigger part of it has to be in the beginning. But as I already said we don't plan it too much into detail because this makes much more sense.

Bilaga 4. Intervju med Richard Morley, OpenBravo.

R1: so have you guys used these projects as well, or are you just interested in the mechanics involved.

O1: Yeah, well since we are informatics students that are our main focus, how you developed the product, and we thought the requirements engineering process is a very interesting aspect because it is important part for the whole development.

[Small talk]

R2: It's an interesting project and I'll tell you why…it's really because it actually goes to the heart, why many people are working in open source…and including my own motivation for working in open source. In propriety solutions or closed solutions, I don't know how you define them…you know propriety commercial solutions...

O2: right…

R3: The process for gathering requirements and converting those requirements into products is very cumbersome and non costumer-friendly. Whereas the open source world is absolutely the opposite, I mean you can pretty much do anything in open source very quickly, and you get a tremendous amount of satisfaction from being able to deliver, you know to define requirements and deliver solutions, very rapidly. Just as an example: in the areas that I'm particularly interested in which is the creating global solutions where you end up with, having to monitor the changing needs of many countries and changing laws in many countries, and as an example, a few years ago there was a change in law in Venezuela that said that all the invoices had to be printed onto pre-printed stationary, so pre-printed, pre-numbered documents which were issued by the government and had to be used when printing invoices. And that add a number of new challenges to ERP systems because the number of lines that in easy invoice was limited, so you could only, you were only able to print nine lines of items on each invoice. So the question was then: what do you do if somebody ordered ten different things, then you have a person that has one order, but you got to automatically split it across two invoices and then you got to handle discount across two invoices. Quite a significant change and quite a challenge for an ERP system to be able to handle that type of requirement. But one which was mandatory for our customers in Venezuela. I defined the need and the business case, we had a significant number, a healthy community in Venezuela. So the business case was certainly there for supporting the legal requirements, but the earliest that I could actually get resources to build it was in eighteen months time, so it really would have taken about two years to get a solution into that market. Basically what they were saying was: we don't have the development resources and we'll just walk away from the market. In Open Source…well the partner was willing to do the development himself, but the propriety solution I was working for at the time wouldn't allow him the tools or access to the code to actually develop the enhancement that were needed and the developers-kit we provided at the time was inadequate to support his needs. So it was just basically impossible for him to make the changes in closed code because the changes that were required went really to the core of many of the ERP systems transactional processes. It really needed to be done by the company, but they decided they would get around to it in eighteen months to two yers time. In the open source world you can address that sort of thing immediately with a local partner, with access to the source code who can make the, who can work with the OpenBravo team and the design, and both the functional requirement, technical design, development, and committing those changes to the core system in such a way that it supports their need but also doesn't have any negative impact on any other user, and you can do that immediately. It is that sort of responsiveness of open source that is very satisfying, and which I think many propriety solutions who claim to be global don't actually acknowledge, they be global by but sort of monolithic and…

O3: right

R4:…and unable to respond quickly, where as open source solutions are able to respond very very quickly and therefore very well suited for exactly this type of application where you have changing local requirements. You know intelligent people who understand their own country's needs and have the skills to develop solution and all they need is a generic ERP platform to build it on, and they're up and running, and that is exactly what OpenBravo aims to provide – a development platform. OpenBravo does not sell its solution directly, it works through an indirect channel which means that we use partners globally, who localize the solution and the partners themselves then sell the add-on services that generate revenue for them. This really responsive model that is very rewarding.

O4: Right, it is a very adaptive way of developing…

R5: Right, it is a very flexible way of responding to local need, local best practice, local legal requirements. It is, not just that, the toolset we provide, because that is what OpenBravo really is. It is a massive developer's toolkit for an ERP system, and the, our objective is to provide the tools that , and the flexibility of the system that partners need in order to localize the product and also build local vertical solutions which differentiates one partner from another. For example in Germany we have a partner building solutions for universities, but we also a partner building a vertical solution in the automotive industry. And they are complementary and what my job is largely is to, in talking with these partners, agree with them what, what they want to do, and which of that is generic and should be part of core system.

O5: Right

R6: And…then OpenBravo then would take that part, you know if I got to partners both wanting to build the same thing, then arguably that, or a large part of what they want to do is very similar, then arguably should be part of the core system. Then they just differentiate themselves in the last mile. I then work with partners to define those common requirements and agree how that common requirement is going to be paid for. What degree of sponsorship and is the partner willing to contribute to have that, have those core changes made. And that appears to work very effectively. It allows us to grow and improve the core of the product in such a way that it becomes a better solution for everybody, and we also work with a principle where if we make a core change or provide a core enhancement, then in parallel we also provide a proof of use of that enhancement. Normally we would do that through the Spanish market, the Spanish market is the only market that OpenBravo actually act as the localizer themselves. So the, for example we built a tool for creating custom tax reports and launching them and it's called, the tax report launcher. It is generically useful, it can be used in any country to create, do tax reporting, you know hard copy and online reporting, integration to online government systems. And all the local partner needs to do is to create the reports themselves, and we provide the Spanish tax reports as templated examples of how you can create the reports to run on that tax launcher. So we provide the generic platform, which is the tax report launcher plus proof of application which is the open source ERP tax reports for Spain, which someone in Germany can open up and have a look at, and customizing change for their own market.

A1: Ok, you mentioned the partners...

R7: yes…

A2: Is that the main source of your …to discover the requirements you later build?

R8: There are a number of sources, the partners and the community are a major source of, not just requirements, but they build themselves. OpenBravo is an open source solutions so the open source community develop solutions. For example: OpenBravo in its instancy could not support an Arabic user interface, the right to left text, also printing right to left documentation and Arabic characters. Now, our partner in Egypt built an add-on that allowed the system to run in an Arabic mode, a right to left mode and then contributed that add-on to core. So it was no longer an add-on, it became a part of the core system. So now OpenBravo seamlessly can switch from western type interface to an Arabic type, you know left to right or right to left, it seamless and inherent in the product. It's a core part of the value proposition that OpenBravo now offers, and that was contributed by a partner, we didn't decide to do that, the partner decided to do it. The partner decided to contribute it. We helped them in, by providing them with guidance on best practice in how to go about doing it and we provided touch points that they needed in the system when they were doing their original, add-on type approach.

A3: Best practice in how to code or how to work with the requirements?

R9: To some degree how to code, but also how to develop it in a way that was not going to cause any disruption to any other user of the system.

A4: ok…

R10: In fact, OpenBravo is completely module based system, it's a modular solution. Even the core of OpenBravo is actually just an enormous module and, you develop OpenBravo in a modular way. One module can not damage any other module, you can create dependencies between modules, for example: let me think, yes, I might create a module which is a tax report, and that module is dependent on the tax launcher module. it needs the tax report launcher module in order to run. It may also be dependent on another module that has configured the applications tax setup. it may also be dependent on another module that contains the tax rates, so you can make a module that has multiple dependencies and it means that it is a very secure and safe way to develop. Because people are working in a modular way, and also we provide an extensive data access layer, we are trying to, well, right all new development is taking place where the modules are talking to core data access layers. That also allows us to make changes to core without actually changing the data access layers and it means that any modules that are developed by partners as add-on, are completely robust and can be migrated from one version of OpenBravo to the next version without any problem. It provides a very smooth upgrade pass for the costumer, an upgrade experience for the costumer that is painless. So when requirements come into core a partner who develops functionality as a number of choices that they can make. They can build a module and contribute it. So basically we say: "here it is, have it, I don't want to maintain it myself but its useful". So that they will contribute it and it becomes, we will either maintain it to the module or include it into core. The partner can, offer that, in alternative the partners can offer that module as an open source contribution, so they maintain the module as a contribution to the community. Or they can license that module as a commercial module, which means they maintain it, but they can actually get maintenance revenue from the costumer for maintaining it. It's a commercial module in an open source world. The core remains open source, but the module that some of the partners build, may or may not be open source and may or may not be free. That's the sort of economic world we're trying to create. But our requirements are really driven, to get back to your original question sorry. Our requirements are driven of what the partners need, but also in a large part by what we believe need to be done strategically to move the product forward and provide a best of breed platform for developers. We want OpenBravo to be a cutting edge tool, so that it is attractive to developers. Those developers always want to work in the latest and greatest so a large part of what we do is working toward making OpenBravo and its toolset cutting edge. So we, a lot of our work in the background is in constantly upgrading and evolving OpenBravo and its platform and its architecture. So that it is that type of cutting edge tool for ease of use and ease of development, easy of customization, ease of localization. So those are, we also have the whole question of usability and workflows, and another large part of what we do is making sure that OpenBravo is as easy to use as possible. But any new development, any new process, any new function, you work smoothly, it's obvious, it's logical, and it's a no brainer how to use it. So the user experience, you know try to delight the end user is also a large part of what we do strategically.

O6: Ok, we just have a couple of questions about the process of working with requirements engineering. If you have any specific requirements process in regards to the core system so to speak.

R11: It depends, the process really depends on what we try to achieve and also which part of the product which I'm trying to enhance. You know what is the objective of the thing I want to build. If we take the, everything we consider to be strategic, for example the, we want to change the user interface to, a different toolset for example. This is something we decide ourselves that there are limitations in our old user interface and we initiate internally a project to research the best toolset for the job. That project results in a both, results in a number of outputs. It results in a functional requirements document, its results in a technical document, but it also results a user experience document. We actually start our requirements gathering frequently from the perspective of what the user will see, what interface, what screen the user will be presented with and the workflows, and we work backwards from there. As an example: currently we're working on a project to streamline the payment flows in OpenBravo. So we basically started with a theoretical payment model that we thought simple, logical, obvious, completely ignoring how the system works at the moment. We started with how do we want this to work [oklart], how do we want it to look, and what do we want it to look like. How do we want to communicate the workflow to the user, and, so our requirements really came from designing a workflow or a user that was simple and obvious and the development of mockup of screens, using graphical design tools to actually mock up the whole process. And then we embarked on a process of taking that mock up into our

community, and demonstrating it both online or live, or to a particular partner and eliciting feedback on that process that we had designed. The visual screens, and people can actually click on it, and click through it and play with it and see how it works, and if we found that people had a problem or if something didn't make sense, they'd provide us with feedback and we would incorporate that feedback into the design. Then we started the, once we had agreed what we wanted to achieve, what we felt would delight our user. We then sat down scratching our heads. How the hell are we going to do it, and, that's the, when the technical team gets involved, the developers really gets involved. It's not to say that they haven't been involved up till then, because in many, they have been involved in our mock ups, because some things we want to do in our mock ups aren't, are simply not possible. So we always try to take a reasonable approach with a light touch from the developers as to, so that all the mock ups are based in reality. But then once we got to the final sort of, process, that we really want to have, and then the developers are faced with the challenge of building it. That's when we move in to our development phase. We use an agile development model where basically we build very, very quickly. Build an demo so a developer might work all night and come in the morning, and say: "Richard, look what I've done, I've done it, I can do this and I can do this after the other, do you like it or not"´, and if I like it, we'll keep it and if I don't like it I'll tell him: "Look its good but this needs to be different, this needs to be different", and they might throw it away and start again. It's this iterative development, this iterative agile development process that then results in the end-product. At the same time we are also in parallel to the development we're also actually doing the technical and functional documentation. So we don't sit down for six months and write huge documents about how this thing is going to work in terms of functional requirements and technical design. We actually build it, test it, road test it, mock it up, road test it, then start designing and building it, and while we're designing and building it, we're creating the documentation at the same time in parallel. So we don't waste any time, and that results in a very, very agile and fast development cycle where the documentation is always in line in reality with what is being built. The other thing is of course is that we don't, I suppose you could say we're quite relaxed about deadlines. We, in the propriety world the deadline for release is everything, you know they're going to release this version on that date, come hell or high water, they will develop and cut scope until they reach some sort of mile stone and final. While in the open source world there is much more flexibility on deadlines. There's a different sort of [oklart], we'll release it when we're ready. And if you know we do have a deadline like a release cycle, then if something is ready then it goes in, if it isn't ready it doesn't go in and nobody panics. It will just come in the next release. So that sort of [oklart] of wanting to get it right and the iterative process until it is right is actually very different from the way that propriety solutions develop, it is a very different culture and a very feeling in the way that the open source product evolve. But then again, that's the only, we're developing our things we feel are strategic, and which feel would benefit our community and where it is important that, because we are a community based product, that we get the buy in from the community on what we're doing. There are other projects that also result in development in core, that are sponsored, these are partner developments, where perhaps there is a deadline. And for those projects, what we do is, we actually tend to ring center a development resource, a development team in the organization and work with the partner on, where the partners defines their functional requirements and then, what we normally will do, in the same process, build and document in parallel, and constantly demo back to the partner, we have development team in OpenBravo, where, you know agile development philosophy, daily scrum, scrum master, are you aware of that?

O7: Right **[**oklart**]**

R12: This is how we work, and …for some of our development projects the scrum master is actually the partner. So the, at the very least the partner participate in the daily scrum, so the partner is very tightly integrated into our development organization, they know, we develop in a glasbulb, everything is visible, we publish and open absolutely everything that we do, the partner can join in, the partner hears our arguments and discussions and problems and challenges, and …but also the partner answers questions and provide input and prioritizes and says Yes and No, and forget about that, it's that important. It means that… it results in very efficient development process, where at the end of the day the partner knows exactly what they are getting, they know exactly when the getting it, and they probably seen it demoed in a number of times, so they don't need to be trained on it. Because they been very closely involve in the development of it, and we then deliver it to an agreed timetable, they then test it and deploy into their model, it might be a vertical model or whatever and that is the sponsored development the deadlines are very important. We bring the partner in themselves, to very close work with us, in terms of the requirements, and the scope and the process. We basically put all washing out on line in public and they see exactly what they're getting. That's another sort of process for the development internal and strategic or whether or something we are doing with a partner with the partner driving the requirements and the

partner actually funding it. And then again we have partners who say: "look at this great idea; I want to build a new evaluation model for stock or something" or "in Finland we evaluate stock in this way" for example, or "we have this decpreciation method", and they develop it. They might contribute it, they contact us in the beginning, we work with them, if the core system needs any adaptation or change to support what they are trying to do, then we will do that without blinking, support the partner's effort to develop and build using our tools. So if we find our tools inefficient then we will extend the tools to support their needs. So when you say, the question: "how do requirements get defined", in the open source world, and certainly in OpenBravo open source world, the requirements are defined by the community and sometimes built by the community, so we try having a community where we are aware what the partners are doing all the time, we are trying, if we realized two partners are doing the same thing then I can put them in touch with each other and make one project out of it. So we act as the controller as well of what the community does and what the community feels needs to be developed, and what the requirements are that the community wants to fulfill. It's a different from a sort of a proprietary, Microsoft type development world, this is a community of developers where we try and make public as much possible what everybody is doing, even ourselves.

A5: Do you involve the end user in this process also, and not only the partners.

R: Yes, we normally work with the partners, but we also, especially in Spain do have a number of. Let me rephrase this, the end users are also part of the community

A6: Yes

R13: And if you, you use Ubuntu or any sort of open source operating system, you are the end user but can still log a bug and log an issue and you can still contribute, you are still contributing and interacting to improve the product even if it is in a quite passive way.

A7: Yes

R14: So, in the same way, we are a community not just of partners but also of users so there are discussions, we actively promote our discussions forum, and we monitor the discussion forum so that we like to see the community answering its own questions. But if a question goes beyond a certain number of days and is not answered, then somebody from the OpenBravo team will jump in and answer that question. So we monitor the discussions in community all the time, and that also results in requirements being defined, because we become aware of the community feels the application is lacking something, which need to be built. And then the community, the discussion goes on about who could build it, which parts should OpenBravo be providing and, so the community actually through discussions and chat and also irc and you know also very public communication channels, blogging. We slowly evolve, sometimes not very slowly, sometimes very rapidly and making a decision, as to whether we are going to do it or not, when we are going to do it, whether it is going on to our roadmap, and then we publish the roadmap. And people are becoming comfortable that, by having these discussions in the community they are actually driving the requirements of OpenBravo and the OpenBravo development roadmap. And we do test other result. We do have costumers, where, for example, my user experience experts. If he blogs about something that, he is doing for example, bank reconciliation screen or bank reconciliation interface. And he, he blog about it and starts a chat and discussion thread. And it's the users that responds, users, partners, I might respond, you know I, I enter to thiese discussions as well in the public forum and give feedback to him and other results he may get an invitation from a particular user to visit their site and actually do a demo there and we do that. He was in Frankfurt a couple a weeks ago, demonstrating a new process that we are developing. You know, we take it to our customers as well whoever is interested in and willing to work with us either in person or through a webinar or whatever. We will interact with that person and take their feedback and consider it very carefully. So really it is development and requirements gathering and setting in a goldfish bulb. Very very public.

O8: Ok, I just have one question about, you mention the agile, I mean the agile development before, you work in iterations.

R15: yes, we work in sprints

O9: Right, and the requirements process is parallel to this? So, or is it, is the requirements process independent, or, I mean do you elicit the requirements all the time?

R16: Well we, We work from what we call our backlog. So, we are continually modifying our, what we call our backlog. And we are continually changing priority of things in our backlog, and our backlog is what the developers eat in to and the objective of the product managing team is to make sure that the backlog reflects our priorities. And there are many factors that may influence how we prioritize a particular development but, there are no mystical science in that. So my team constantly changes and adjust the backlog. And this is, the backlog is what the developers then eat in to in through these sprints. The backlog is allocated in different teams, the teams work in sprints, they, the team decide how long, they look at the backlog for them, the backlog allocated to their team, they then sit down and decide how long the next development sprint is going to be. How much they can take on, what they will do, and they agree that with me or with the partners, whatever. And then they go and build it. And they tend to build in a iterative way in the sprints, they might build it in one day and show me and I say great, move on to next thing, its fantastic, I love it, but normally they work in a iterative process where they work as a team and in the sprint and they commit to delivering a certain part of the backlog, they consume that backlog, eat in to it. What I ask from them at the beginning of the sprint is what can they commit to delivering at the end of the sprint. And then come hell or high water, they have to deliver on that. I don't care how they do it, as long it meets my, satisfy my requirements. You know, if they have to work day and night to deliver it, they deliver it. But they, the fact is then, as they get more experience, what happens is then they become very experienced in estimating the effort and breaking down the tasks, that need to be done to deliver anything. And it means that we then have very accurate estimates of effort and, it means that the sprints are actually normally pretty accurate on how much effort it is going to take to deliver a particular thing and everybody is happy. Because I get what I, I get the commitment to deliver, and then they deliver. On their side, the developer side, they trust that I'm taking their work for it about how long it is going to take to build something. They feel that they can do and they do, do it so they get a stance of accomplishment. So the developer are constantly eating into my backlog, which I maintain, the backlog, when I present it publicly, I call that the roadmap. When I present it to consumers or partners or whoever. The roadmap is, almost like a summary of the backlog, it's the themes of the backlog, and it's the major milestones of the backlog, by June next year I intent to be at about this point in the backlog. And there are key and milestones in there. So what we call the backlog internally is repackaged as the roadmap for public consumption. Even the backlog itself is public but the backlog is constantly reviewed, and we manage it through what we call a scrum of scrums. Because we have multiple of scrum team, but once a week representatives of all of the scrum teams get together and have what they call a scrum of scrums, where they all touch base where they are in different parts of the backlog. Because some things may, one feature or part of the backlog may depend on another feature that another team is working on. So the scrum of scrums are to coordinate what the other scrums are doing, in order for them to, as one machine eat through my backlog. Does that make sense?

O10: Yeah, kind of. Quite complex though.

R17: Its not really, the backlog is consumed by the developers, and the developers work in scrum, collectively all the scrums are manage by the scrum of scrums, its one big scrum, well, not one big scrum, its representative from of each of the different scrum teams, and they get together and make sure that everybody is aligned, for example, as an example: I have one team developing financial applications, financial functionality, I have another team developing platform type functionality, in one **[**oklart**]** the financial team actually uncovered a bug in the platform, and it needed the platform team to respond to respond in their current scrum to fix that show stopping bug for the financials team, so the scrum of scrum resolved that. They **[**oklart**]** from the financial scrum, **[**oklart**]** .. and the scrum of scrums agreed that the platform scrum team needed to adjust the scope of their current sprint and deliver a fix to the other scrum team, and they did that. And that's the role of scrum of scrums, optimize the development and optimize communication between the scrum teams.

O11: Ok, we have a question about the reuse of requirements, does it happens that you, you have this backlog, I guess you could define it as some , if you don't implement some stuff you go back and look at it later and you can, you understand my question?

R18: That's right, stuff on the backlog is the priorities juggle, so it might be there, strategically I plan to do something with a certain scrum team, but then a partner approach me and put money on the table and say "look, I urgently need this functionality" so that may then change, I may decide to take that money and work with that partner and reallocate the resources to supporting that partner. And that means that my, that the priority of something I have planed to do changes and drops down in the backlog. I move it down in the backlog and insert something higher up that I decide to do. So something can stay on the

backlog, for a long time actually, the backlog can be translated into a roadmap of say, one to two years of development. But, its volatile, it is constantly changing. In terms of requirements, all requirements that we get, all requirements have to come from someone. There has to be a sponsor of the requirements, either a partner or costumer, or somebody internal in our own organization that commits to providing the functional requirements. What my team, the product managing team does, is work with that sponsor, to help them develop, define the requirements and we also may **[**oklart**]** the requirements, we might work with the sponsor of the requirement to agree to what the priority are in their requirements. There might be multiple deliverables, and we say, what is phase one, what do you want first, can we put this to phase two or phase three? And we agree how those requirements are going to be, the importance of the requirements, what they should be, and when they are actually required. So we work very closely with the sponsor, whether that is a customer, or a partner, or whoever. And defining and making sure the functional requirements are as clearly documented as we can. But to be honest, we are also far more relaxed perhaps, about the functional requirement than the properties solutions, because we a very respectful of the fact that the requirements change**,** through understanding or just time or something. So our requirements, we do take a very flexible approach to requirements but at the end of the day the sponsor of the feature, the person how actually has defined the requirements, at the end of the day the get actually what has been agreed with the team as being the best solution. They always get the best solution and they know they get the best solution because they worked so closely with the team in defining in those requirements , how its going to work, how much time its going to deliver, in the proprietary world, what tends to happens is that the costumer, say "I need this after the other" and the proprietary erp solution provider thinks well I know how to do that, I know best, they design their own solution and effectively impose it or unveil it at the end of the day and train the costumer how to use it and fix any issues that exists. It's a black box type approach, we are not a black box in any way, we are a completely transparent goldfish bulb.

O12: Right.

R19: One of your questions was change management.

O13: Right, you said you had a flexible approach.

R20: Yeah, I just want to tell you a little bit about change management, because we are flexible in the requirements and the whole requirements definition process. And we have a very interactive approach in defining the requirements for our customers, and while that is going on, any change is permissible, until the point that we start development, at that point it depends who we are building for, because if we are building for ourselves or something strategic, then we manage change on a consensual basis, basically we agree or disagree that a change needs to be made, and the reason we can do that, or shouldn't be made, and the reason we can do that is because we have far less pressure on deadline, we are just very comfortable, we deliver when we feel it is ready,

O14: Right

R21: However, if we are working on a project where there is a partner involved and a deadline. What we do is we initiate a requirements gathering process, where the functional requirements are agreed with the partner. We then enter into a development stage, and at that point, we create a project, a development project and the requirements are published into that project as a static document, and that document is also associated with its own discussion forum, so if the developer has a question while they are developing they ask that question in the forum for that project. And that creates in the document, in the requirement document, an open issue. Then the open issue links through to the discussion thread in the forum.

O15: Okay

R22: Now, the, sponsor, the partner or me, or whoever who is driving the requirements, then provides feedback to the developer to answer their question. And the developer responds with an explanation, in the forum, in the public forum of the impact on the project, will it delay the project, or shorten the project, basically an explanation about what the impact will be, of a particular change or a change of mind or maybe something we didn't realize or nobody realized, but it is a change. We in the forum, publish what the impact on the project will be. The sponsor, then have to accept or reject, basically they say "Ok, I'm fine with this change" At this point the discussion thread is used to update the functional requirements so the functionality that was agreed on the forum is then actually transferred into the functional requirements

document. The project delivery date is updated with the information from the discussion thread in the forum and the open issue is moved to a closed issue, so that means at the end of the project we have the original document, the original functional requirements, and we have the final functional requirements. And also, at the end of project a list of closed issues, and for each and every closed issue a breakdown of what impact that issue had on the project and the explicit acceptance or rejection of that issue. An open issue can be rejected and become closed, or it can be accepted and become closed. But basically you have this, you have a full delta between what was we original asked for, what we delivered, and why we delivered that and what it cost in terms of time, resources etcetera and therefore you have complete change control between the original requirements and what was actually delivered. And also complete acceptance by the costumer, so we could say that all our collaborative development projects where we work with our partners, we could say that all of them are delivered as agreed with the partner. We never deliver something that is not agreed. Because it just can't happen using this methodology, they always have to either accept or reject the change. Because that's the reality of life, and the reality of projects. So we never. When working collaborative with a partner, we always deliver on time. Because of our change control mechanism, I mean that time may be different from the original plan time, but it's certainly the agreed time at the end of the project.

[Tilläggsfrågor till intervjun]

O16: We just have some specific questions regarding the analysis of requirements, specifically how you find problems with requirements. When you have discovered a requirement, how do you find problems with this requirement, like ambiguity or conflicts with other requirements or incomplete requirement and so on.

R23: Well it depends what the requirement is, I suppose and I'm not quite sure of what you mean by problems, what sort of problems are you alluding to.

O17: Well, for example if you already implemented a sort of similar requirement into the core system like a feature that is sort of similar to a new requirement that has come up? Or if the requirement that you have is sort of hard to interpret…

R24: Well we don't start development until we know what we're doing, so we always go through the process of at least agreeing a high level requirements statement or document whoever the sponsor is of the feature. Whether that is an internal sponsor or an external sponsor, we always start up with a pretty good idea of what they want, and if we think that existing feature simply needs to be extended then that's what we will do and a lot of that comes down to our own knowledge or understanding of the product. Just as an example we developed a solution for automating intercompany invoices and then our partners approached us wanting a solution for internal invoices between internal departments of one organization and they wanted the transactions to be held, or to be removed or not included in the balance sheet in the end. Because these transactions don't contribute [oklart] loss, they're just simply internal control purposes. So what we did was we took the intercompany functionality and we extended it with a new document-type for internal transactions, and we also then extended the reporting solutions so that you could chose to include or exclude any particular document type or transaction type, and that allows you to exclude internal or transactions derived from internal documents. So that was an extension of, an extension to an existing module that we had built. So we built a module for intercompany and we extended it to support intra organizational transactions. What we don't know if we're developing a solution, and at the same time a partner might be developing a similar solution and we don't always know that. But the architecture of OpenBravo, it supports this, the way that all development is done is through a data access layer, so there isn't any possibility for modules themselves to conflict. We can load them for example and test them, and obviously make sure that everything is working fine. The problems we have, our partners is the other way around, are not conflicts, that we have to manage, the bigger problems is dependencies, and it might be that one module depends on other modules and it's important that the system is always in a consistent state and continues to run. That it itself is also an engineering exercise, and not something that generate huge problems. I think it's a lot to do with the platform, and the OpenBravo platform, because it is a modular platform allows for modular development without conflict. It's a fantastic tool for exactly these problems. I don't know if I answered your question.

O18: Yeah, that pretty much answered our question. I guess you kind of answered this one too actually. How do you make sure that the requirements you have specified are the ones that the stakeholder wants. Before you told us when you work with a client you have discussions and so on. But how do you do when you have requirements from users and so on.

R25: Let me give you a link, let me give you an example of how we expose concepts and ideas.

[Richard lägger upp länkar i Skype-chatten]

R26: So what we're doing here…Let me see if I can find Rob's blogg as an example. This is the whole, the last one I sent you is the whole redesign of the user interface. So what we do is we start talking about it, we start blogging about it, what we want to do, the ideas, the concepts, you can see those videos we've created on the "concepts and financials". If you click on the OpenBravo.com concept financials link, it's the first that I sent you. You'll see that it's an invitation for feedback and we, we've got a new, we're revising the sort of whole payment flow process in OpenBravo, where you can watch these videos. They're just mock ups of what we want to do, and you can give feedback on the user experience labs forum which is actually the second link that I just sent you, and another third link I just sent you is us generating is about new user-experience, and again demo-videos is the third link, and the concepts for OpenBravo ERP redesign. There's the demo, the forums, there's surveys for each of the main features, so if you click on the third link you'll see on the bottom of that page: "please fill out the surveys about master detail, about the search and filter functionality, about the concepts about myworkspace and overall impressions", so we engage with our users and our partners like this, this is how we do it. We talk about it, we do some design work, we start blogging and mocking up and publishing and generating discussion and basically getting feedback and then we filter that and that gets converted to a requirements document. In parallel technical design, the redesign, all this stuff is, it would be wrong to look at this as a linear process. When we're doing the mock ups, we're also thinking about technical design, we're also documenting functional requirements, it's sort of, it's a multithreaded cyclical process that we've, if things change we just change direction of the whole ship as it were and work that way. I think these examples are pretty great of how we do it. You'll find blogging is a big deal for OpenBravo so have a look around the OpenBravo blogs and see what's been spoken about see what sort of know the company generates and what sort of feedback the company tries to elicit. These two examples I've just given you, the financial process and the UI, they are sort of the two big things were working at the moment, and I think they're quite good examples of how we elicit feedback and making sure the requirements that we envisage are actually correct and appropriate. We also do demos, these videos that we have here are actually recordings from what would have been live sessions of what we've conducted with the partners and costumers, we have a small group of users and partner we know are competent in giving feedback and they're skilled, we know they understand what it is we're trying to do. So we normally try to elicit feedback from them as well and make sure we get their views, and that's basically how do it.

O18: Do you use blogging for all your development that is related to the core-system?

R27: Yes…Gonna send this to you…Paolo…Paolo is our chief-technology officer, and I think, like all of us, he's quite a keen blogger and I think that you can see pretty much what we blog about and what we look for feedback on, it's not just asking for feedback, it's also giving updates where we are, what we're doing. Like I said, everything is in a goldfishbulb for us. Have a look through that, because there's quite a lot there to, I think reading Paolos blog will give you quite a good flavor of how we interact with our communities so, and also maybe the [oklart] as well, modularity is a big thing with us at the moment, so he's got a number of blogs about modularity. You can search that on the label of modularity. You can search Paolo's blog for modularity, that's what I'm talking about when I tell you that OpenBravo is built in a modular way, and conflicts just doesn't happen because of the architecture. So if you read [oklart] about modularity, I think you don't need to read about the technology, just read about what Paolo is blogging about it, it will give you a flavor of how it works and why we as an open source company have this really strong technical platform, that is ideal for community development and stand-alone modular development.

O19: This is great, thank you Richard.

R28: I think it's worth having a quick look through it because the modularity is one of the corner stones of development, everything we build internally are built as modules, everything our partners build we expect to be built as modules. Except the only caviac is that modularity is available from OpenBravo 2.50 and a lot of the partners are still working with all the versions. But we're helping them migrate all of their development as modules on to 2.50. But have a look through Paolos blogs, that will give you pretty good ideas of how we interact with our community. The blogs from Rob is user-experience and it's, a large part of how we design these days is, ok given an ideal sceniaro, how do we think it should work and we test the ideal situation with mock ups and once we got feedback on that we build it…the module.

[Smalltalk]

O20: A question about the backlog, the requirements that you put in your backlog, they have gone through this, kind of process you just described to us. Or you have discussed them with you clients or partners.

R29: Yes, the backlog has three main pillars to it. That we name pillars. One is that we want to maintain our competitive edge and also that with our partners. So we want to maintain and continually develop the tool that we provide to our partners. So in that sense, a lot of what we do to maintain the product is actually strategic development that we decide is critical to maintain our competitive advantage and to make OpenBravo attractive to people that are deploying it. The other theme we address is the [oklart] pillar of our backlog is the what we call it delight. That is we want to develop features and functionality that would delight our users, users in terms of end users and also users of our tools and users of our development environment. So many features that come into the delights category have actually come from our customers or come from our partners. So we go through our issue tracking system and then we look through the feature request and we discuss them internally and rank them. And that sort of governs their position in the backlog. And the third pillar that we normally use for decision making is monetize. These are features that allow OpenBravo and its partners to make money. These might me license features, they might be infrastructure features for example a trading platform or modules is something we building right now. So that is a platform to allow partners who have build modules to effectively sell them. Sell them to end users or sell them to other partners. So that in exchange, an module exchange. So a tool to help partners to make money, to monetize. And that also, lets not beat about bush here, we are in the business to make money. Our end users and the users of our solution, expect us to make money and would be worried if didn't make money. Because nobody want to be using a mission critical solution that is built by an organization that doesn't maintained by an organization that isn't profitable. So we want to make money, we want to be able to demonstrate to our partners and end users that we are a financial stable organization and we also want to demonstrate to our partners that the scope within OpenBravo world to have a profitable business servicing the OpenBravo community. And for that symbiotic relationship to evolve that's why we consider monetize as our pillar. So its maintain, monetize and delight, these things are the things we look for in our backlog of possible features and we prioritize them along those. And we also do publish our backlog and ask our community to rank it

[Richard letar upp en länk till backlog]

O21: Thats great, we were looking for it before but we didnt find it actually so, all we found was some documentation in you wiki or whatever you call it, but we didn't actually find the backlog.

R30: The backlog we keep internally, what we publish is the roadmap. The roadmap is the summary of the backlog. In fact we are actually overdue to publish a roadmap, so we are going to publish a roadmap in the next couple of weeks.

[Richard letar vidare]

R31: That takes you to, what is effectively highlighted is this page here. So this was the last time we did it, in march so, and that, we are starting a next round of development so we review this, this one of many tools that we use to trying to rank how people are, how people in the community see gaps or [oklart] enhancement in the product., the user voice [oklart]

O22: This is really interesting, so you put up suggestions for example or changes that, people have requested or that you…

R32: People log into, you see that people have reported an issue and we have classified that as a enhancement request rather than, it is not an issue, we don't think it is a bug, we just think of it as a nice to have feature, functional enhancement. Or people have send us email, or we have been at conferences, and people have discussed it or we look forward and think that strategically it would be a good thing to do. And just [oklart] we uses this time of survey where we actually ask people what they think.

Bilaga 5. Intervju med Si Chen, Opensource Strategies

O1: Can you shortly tell us how you elicit and discover requirements and so on?

S1: There's a core of OpenTaps, which supplies the framework and the core features of an ERP and CRM system and over time additional features are added on to it. The additional which show up, a lot of them are developed either by ourselves or clients, or they are developed by other people, either from client projects or because they wanted some enhancements in OpenTaps and contributed that into OpenTaps. The process of requirements gathering in that case is pretty similar to any sort of custom developed feature for a specific group of users. Then basically we would go to the user and discuss exactly what they're looking for, and we would put it usually on an issue tracker to describe what their requirements are. And what different scenarios are and check that with them and then we would develop the feature and they would get a chance to check it again. It's pretty standard methodology of requirements checking, and I guess how it's documented, the way that requirements are sort of preserved in this system is through documentation and automated testing. So basically we a have documentation wiki, where you can add entries on how different pages work and how different features work. So for example if we added a button or a field on a form, we would go to the corresponding page on the wiki and update that so that the page: "This field does such and such". Then more importantly, where it's feasible and where the features is important enough, we would have automated tests for it. We would basically write a unit test that covers all the scenarios of the requirements. They're really not unit tests in the classic sense, they're more functional tests even though we use the J-unit libraries for doing this. So the requirement would be for example contained in an automated test rutine that might say something like: "create a product" and then "create a costumer" and "create a order for this costumer" and process the order and make sure those invoices are created and that test basically keeps that requirement around in the system, and we run the test regularly and we accumulate tests over time. So basically by making sure that these tests continue to work while the software is developed, we make sure that those requirements are preserved. So I guess that is basically how it works, if you have any questions more specifically I will try to answer them.

O2: What's the main source of requirements?

S2: I guess it would be from users of the system and people, a lot of the functional features come as a result of projects that users have requested. Then we invest a lot of time and resources into building up the core system so that there is a user friendly well-architected core-system, where we could develop these additional functional features easily. So for example we invest a tremendous amount of time into building a user-interface and the AJAX-user interface in OpenTaps and then somebody would come and say: "We would like a screen that would do such and such a thing" and we would build it with that, the user interface framework we have developed. It depends what you view as the requirements of the system or what you view as requirements. A lot of the real value is not in the feature it has, but really in the underlying architecture. In face I would argue that the real value is that, not the actual screens and forms and things that it does.

O3: Are there any other sources of requirements except for the users?

S3: Well some features are generated from community feedback and features developed by people in the community, so those tend to show up though, more or less evolved. You know what I'm saying, like there is a set of features or screens that show up from a contribution made by people and so we would typically document that as well for them before we implement it into OpenTaps and a lot of times we would also write automated tests for them. An alternative is we just encourage people to just have it as an independent module for OpenTaps, that's not part of the core project and then they're responsible for maintaining, gathering requirements and furthering it's development, and then don't forget for example we use this system ourselves for a variety of things and so do the some of the other developers of the

system, so they contribute also to own development. They are their own requirements and their own users, but I think a little less formal, I don't create a whole set of documents for my own features and while I'm thinking about some new things that needs to be done for the system. But I guess I follow them into process, but not as formally. Requirements have to, they come from the user and they should come from the user, because that's how you know if someone really needs the feature and how it should work.

O4: How do you involve the stakeholders?

S4: We will show them, it's basically the same process I think as in engineering of commercial system. You would show them different features, and they would tell you what you need, and then they would, you would look at different options and start developling the feature together and sign off on it and then we would add tests and documentation for it. I guess what's interesting also though is that requirements in an open source project like OpenTaps they sort of evolve in small iterative steps and they accumulate more over time in an open fashion, so by that what I mean is there could be an additional feature set which then one group of user come in and say we would like it work this way, and so we might extend it for them and then they might, and then we make it a part of the open source project, we documented it, put automated tests in place and then later on somebody else could come in extensions and additions to the features to the first set of changes we've made. Either they would contribute something or we would add enhancements to it and then we would extend the documentation and the automated tests. So basically what happens is that, as you see a feature evolve it's absorbed several sets of requirement but not necessary from the same group of users. Do you see what I'm saying.

O5: Yes

S5: Right, so for example, one of you could come to us and say "we want a feature for doing x" and then we would make x1 which incorporate all of your features plus documentation automated tests. And then the second person come in and say "we want a additional set of features" so then x1 becomes x2 with documentation and automated tests. But, generally the features in x1 are still preserved. And they are preserved and they are preserved precisely because there are documentation and automated tests which sort of, set the original requirements, into, I don't want to say into stone, but they set in into a formal and maintainable form.

O6: Right. Can we ask, do you reuse requirements from other projects?

S6: Do we reuse requirements from other projects? You mean like go look at another open source software and see if they have good features we can implement?

O7: Or rather from you own projects or other features that you planned to implement before but maybe you didn't implement a specific requirement for that feature.

S7: You mean like for example if we implement a feature but then we leave some, like there is some unimplemented features and then we implement later?

O8: Yeah.

S8: Could you give me an example of this?

O9: Well, lets say you have a requirement for a special feature from a user or something and you process this requirements. But you decide that due to time constraints or resources or some other reason. That you don't want to go ahead, but then later or you look back and you decide that it is worth implemented.

S9: Yeah, that's happen more, that does happen from more, I think, in the community for example. Sometimes people will suggest different things and they will ask for different features, so they are not really client projects or something that we have mandate to work on in a timeframe to deliver to people. And sometimes the ideas are very good and we can implement them or sometimes it takes us sometime to get around to it. But if it is something make sense we would definitely act on it. But maybe not right away.

O10: That's interesting. Can I ask, lets say, between different client project, does it happen that you look at a thing you have done with another client and they have come with a requirement for the same reason

and then you take the requirement from that project or that work with the client and use with another client?

S10: Only if the original work becomes part of the open source project, so for example if somebody ask us to implement enhancement and then we put it into the open source version on OpenTaps then someone else might pick up on it and ask for some addition enhancement. My x1 and x2 example from before, you know where the same feature with different requirements from different people passed on over time. But if it is not put into the open source requirement then no, there is not much overlap between the work that is done for one client to the other, they tend to pretty separate things that is done for different people.

O11: Right. Do you prioritize between requirements?

S11: That is a good question. Do we prioritize? Yeah I mean I think client projects that we have taken on have obviously high priority, because we have a mandate to work on them and also there is a time issue. And then, so it is professional services and support client projects have a higher priority. And then we actually prioritize, based on whether they would benefit the largest numbers of our users. So for example, a lot of people will come to us and say " you know we would like this" and that, if it is something that we think will benefit a lot of users we will give it a higher priority. But if it is something that only a few users or a very specialized feature that a few users would want, that becomes a lower priority for us.

A1: But in the client projects, who prioritize the requirements there, is it you or is it the client?

S12: Well, each client prioritizes their projects. And our goal is to get all our client projects done in a way that is satisfactory to them or else we wouldn't take them on, obviously. We don't want to get in a situation that we take on projects we can't do.

O12: I think you kind of answered before, but how do you respond to change in the requirements? Do you have any special way of dealing with that?

S13: Lets see, change in requirements. We basically , well you mean from the same group of people or over time as different requirements show up?

O13: Well, both actually

S14: Yeah, I think we already discussed some of this, but from. From the change in requirements, so if there is a, if we are working on a feature set, lets go back to the original example I gave you, there is feature set x and then you come and tell us you want feature set x1 and then some point through there is x1prime and you want us to change it, to that. Your requirements has change and this is not unusual, I think we would just modify the code or feature set until it is where to you where you needed to be. And I think we have to judge whether that it is still something that make sense to the open source part of OpenTaps or whether that it becomes something so specialized that it only work for one group of users, basically yourself. But assuming that it still works for OpenTaps we would incorporate x1prime instead of x1 in the core, in the OpenTaps system. And then if you say that there is additional features, that just basically, it is just get added on top of the original feature set and more tests and documents are in. Nothing to special there I think. It is just an iterative process of the system being used of people and shown to people and their feedback gets incorporated and the system is modified.

O14: A question about, after you have, after you got an requirement or started out working with a client and they come with a couple of requirements. And you look them over, how do you decide when the requirements, how do you decide when they are complete. I know you work in a iterative way but, how do you decide when you start working with them.

S15: Whether, that the requirements are completed? I guess they would tell us or we would also have our opinion of whether the whole system work well as a whole together. So for example they might say "Ok this feature does everything we need it to do" and then, we often would also invest some more time to make sure that it works well with the rest of the OpenTaps system if it is going to be part of OpenTaps.

O15: How do you do this? Do you discuss it or do you have a special technique?

S16: No special technique, we just discuss it with the client and they go over it with us and they basically us "this and this looks good and works for us" and then we are looking on at it from the whole angle of

the rest of OpenTaps, we are relying on our experience with the system but also feedback from other people who are in the community using OpenTaps, they may have questions and issues make us think about how it will fit into OpenTaps.

[Tilläggsfrågor till intervjun]

O16: You said: "Then basically we would go to the user and discuss exactly what they're looking for, and we would put it usually on an issue tracker to describe what their requirements are and what different scenarios are and check that with them and then we would develop the feature and they would get a chance to check it again." Could you describe how you check the elicited requirements with them?

S17: Typically, we would review the existing functionality with the user first, and then discuss changes or additions they are looking for as scenarios.

O17: You said "We will show them, it's basically the same process I think as in engineering of commercial system. You would show them different features, and they would tell you what you need, and then they would, you would look at different options and start developling the feature together and sign off on it and then we would add tests and documentation for it." How do you show them? Do you apply any special technique(-s) doing this ?

S18: For features developed for specific users, it would be typically in person or remote desktop sharing meetings looking at the same screens together. For features developed for the community, we would sometimes describe it in the general forum, or sometimes develop a prototype and then elicit feedback.

O18: And who do you show? (clients or their users or both or any other stakeholders?)

S19: Most of the time just to specific users were interested in the feature. However, since opentaps is an open source project, it is a "living project," so as features appear in the project, other people will give us your feedback and enhancements over time.

O19: From the interview we got an ambiguous picture of the documentation within your requirements engineering. Do you document during the time of the requirements process and for what purpose, or do you only document after you have implemented a feature?

S20: Usually after the feature has been implemented, for the purpose of allowing other people to use the feature and also documenting the requirements for later use.

O20: Do you reuse requirements from internal projects within your own company or requirements from development of earlier features?

S21: I'm not sure it we do exactly that, but as opentaps is "a living project" requirements are constantly being reused and recycled. In this way it's a bit like a city: each building in the city represents the requirements of its residents at one point, and the buildings are reused and modified over time to meet new requirements. So it is the same with opentaps.

Referenser

Ahmad. S. (2008). Negotiation in the Requirements Elicitation and Analysis Process. *19th Australian Conference on Software Engineering (aswec 2008)*. [Online]

Bahill. A, T. & Henderson, S, J. (2005). Requirements development, verification, and validation exhibited in famous failures. *Systems Engineering*. [Online] 8(1)

Bjerre. L. (2008). *Affärssystemet – en livsnödvändighet.* [Online] IDG. Tillgänglig på: http://www.idg.se/2.1085/1.156198. [hämtad 21:e november 2009].

Boehm. B.W. (1984). Verifying and Validation Software Requirements and Design Specifications. *IEEE Software.* [Online] 1(1)

Daneva. M. (2007). Understanding Success and Failure Profiles of ERP Requirements Engineering: an Empirical Study. *33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO 2007).* [Online].

Davis. A.M. (1993). *Software requirements: objects, functions, and states*. Rev. ed. Upper Saddle River: Prentice Hall PTR.

Davis. A.M. (2005). *Just enough requirements management: where software development meets marketing.* New York: Dorset House Publishing.

Decker. B. et al. (2007). Wiki-Based Stakeholder Participation in Requirements Engineering. *IEEE Software.* [Online] 24(2)

Dorfman. M. (1990). System and Software Requirements Engineering. I R.H. Thayer & M. Dorfman (Red.), *System and software requirements engineering: IEEE computer society press tutorial* (s. 4-16). Washington: IEEE Computer Society Press

E-Delegationen. (2009) *Tredje generationens e-förvaltning.* (Statens offentliga utredningar 2009:86) Stockholm: E-Delegationen [Online]

Eriksson. U. (2008) *Kravhantering för IT-system.* 2nd ed. Lund: Studentlitteratur.

Fitzgerald. B. (2006). The Transformation of Open Source Software. *MIS Quarterly.* [Online] 30(3)

Grünbacher. P. & Seyff. N. (2005). Requirements negotiation. I A. Aurum & C. Wohlin (Red.) *Engineering and managing software requirements.* (s. 143-162) Berlin: Springer

Hickey. A.M. & Davis. A.M. (2003). Elicitation technique selection: how do experts do it? *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International.*[Online]

Hull. E., Jackson. K. & Dick. J. (2005). *Requirements Engineering*. 2nd ed. [Online] Berlin: Springer-Verlag. Tillgänglig: http://www.springerlink.com/content/gwl591/. [hämtad 7:e oktober 2009].

IEEE Computer Society. (1998). *IEEE recommended practice for software requirements specifications*. [Online] IEEE. Tillgänglig på: http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&isnumber=15571&arnumber=720574&punumber=5841. [hämtad 22:e november 2009].

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

Jiang. L., Eberlein. A. & Far, B.H. (2005). Combining Requirements Engineering Techniques – Theory and Case Study. *Engineering of Computer-Based Systems, 2005. ECBS '05. 12th IEEE International Conference and Workshops on the.* [Online].

Karlsson, J. (2006). Software requirements prioritizing. *Requirements Engineering, 1996., Proceedings of the Second International Conference on.* [Online].

Kotonya. G. & Sommerville. I. (1998). *Requirements engineering. Processes and techniques.* Chichester: John Wiley & Sons.

Lemos, R. (2008). *Open Source ERP grows up.* [Online] CIO. Tillgänglig på: http://www.cio.com/article/339321/Open_Source_ERP_Grows_Up?page=1&taxonomyId=1461. [hämtad 6:e oktober 2009].

Loucopoulos, P. & Karakostas, V. (1995). *System requirements engineering*. Berkshire: McGraw-Hill Book Company Europe.

Macaulay. L. A. (1996). *Requirements Engineering.* London: Springer-Verlag.

Magnusson. J. & Olsson, B. (2008). *Affärssystem.* 2nd ed. Lund: Studentlitteratur.

Rapp. J. (2009). *Ökat intresse för Open Source.* [Online] Logica. Tillgänglig på: http://www.logica.se/r/400016853/%C3%96kat+intresse+f%C3%B6r+Open+Source/400016365. [hämtad 6:e oktober 2009].

Robertson. S. & Robertson. J. (2006). *Mastering the requirements process.* 2nd ed. Upper Saddle River: Addison-Wesley.

Robson. C. (2002). *Real world research.* 2nd ed. Malden: Blackwell Publishing.

The Standish Group. (1995). *Report CHAOS.* [Online] Tillgänglig på: http://www.projectsmart.co.uk/docs/chaos-report.pdf. [hämtad 6:e oktober 2009].

Sailor, J.D. (1990). System Engineering: An introduction. I R.H. Thayer & M. Dorfman (Red.), *System and software requirements engineering: IEEE computer society press tutorial* (s. 35-47). Washington: IEEE Computer Society Press

Sillitti. A. & Succi. G. (2005). Requirements engineering for agile methods. I A. Aurum & C. Wohlin (Red.) *Engineering and managing software requirements.* (s. 309-326) Berlin: Springer

Sommerville. I. (2007). *Software Engineering.* 8th ed. Harlow: Pearson Education Limited.

Sommerville. I. & Sawyer. P. (1997). *Requirements engineering: a good practice guide.* Chichester: John Wiley & Sons Ltd.

Statskontoret. (2003). *Öppen programvara.* (Publikationsnummer 2003:8) Stockholm: Statskontoret. [Online]

Verville. J.J. Palanisamy. R. Bernadas. C., & Halingten, A. (2007). ERP Acquisition Planning: A Critical Dimension for Making the Right Choice. *Long Range Planning.* [Online] 40(1)

Wiegers. K. E. (1999). *Software Requirements*. Redmond: Microsoft Press.

Wärneryd. B. et al. (1993). *Att fråga: Om frågekonstruktion vid intervjuundersökningar och postenkäter.* 5th ed. Örebro: SCB-Tryck

Young. R.R. (2001) *Effective Requirements Practices.* Boston: Addison-Wesley.

Zhao. L. & Elbaum. S. (2003). Quality assurance under the open source development model. *The Journal of Systems and Software*. [Online] 66(1)

Zowghi. D. & Coulin. C. (2005). Requirements Elicitation: A survey of techniques, approaches and tools. I A. Aurum & C. Wohlin (Red.) *Engineering and managing software requirements.* (s. 19-46) Berlin: Springer