

LUT

– EN METOD FÖR SYSTEMUTVECKLING MED LEAN SOFTWARE DEVELOPMENT

Kandidatuppsats i Informatik

Patrik Björklund

VT 2011:KANI07

Svensk titel: LUT – En metod för systemutveckling med lean software development

Engelsk titel: LUT – A method for system development with lean software development

Utgivningsår: 2011

Författare: Patrik Björklund

Handledare: Stefan Cronholm, professor

Abstract:

Traditional systems development is often plagued by a long process with an inability to meet changing customer requirements. Problems with bugs in software and moved deadlines are recurring problems that lead to higher costs for companies.

To be able to deal with these problems a systems development process is needed that identifies problems at the moment of creation and forces the involved parties to reflect over the process that created the problems as well as the real reasons behind the problem. Low quality in software is often a symptom of deep organizational problems.

One way of thinking that aims to meet these problems is *lean software development*. The view is inspired by *lean production* which is a description of The Toyota Production System which in turn is the process that Toyota have used for decades in traditional manufacturing.

Mary and Tom Poppendieck have described seven principles and 22 thought-tools that can be used to create a method for *lean software development*. This study shows what strengths and problems that *lean software development* and *lean production* describes in general and what strengths and problems exists in the description of Poppendieck & Poppendieck's principles and thought-tools specifically. This shows what improvements needs to be done to be able to use the principles and thought-tools as a base for a method for *lean software development* and how this method would look concretely.

The studies result is a method for *lean software development* based on Poppendieck & Poppendieck's principles and thought-tools by the name of LUT – Lean systemUTveckling.

The thesis is written in Swedish.

Keywords: lean software development, systems development, method

Sammanfattning

Traditionell systemutveckling präglas ofta av långa utvecklingsprocesser och en oförmåga att möta förändrade behov från kunden. Problematik med buggar i mjukvaran och förskjutna deadlines uppstår kontinuerligt och leder till högre kostnader för företag.

För att komma till rätta med dessa problem behövs en systemutvecklingsprocess som belyser problem direkt när de uppstår och därmed tvingar de inblandade till reflektion över processen som skapar problemen vilket belyser de verkliga bakomliggande orsakerna. Låg kvalitet på mjukvara är ofta ett symptom på djupare organisatoriska problem

Ett synsätt som syftar till att möta dessa problem är *lean software development*. Synsättet har inspirerats av *lean production* som är en beskrivning av The Toyota Production system vilket i sin tur är det arbetssätt som har tillämpats av Toyota under flera decennier i den traditionella tillverkningen.

Mary och Tom Poppendieck har beskrivit sju principer och 22 tankeverktyg som kan användas för att skapa en metod för *lean software development*. Studien visar vilka styrkor och problem som *lean software development* och *lean production* beskriver generellt samt vilka styrkor och problem som finns avseende beskrivning av Poppendieck's principer och tankeverktyg i synnerhet. Detta visar den förbättring som behöver göras av principerna och tankeverktygen för att de ska kunna användas som underlag för en metod för *lean software development* samt hur denna metod ser ut konkret.

Studiens resultat är en metod för *lean software development* baserad på Poppendieck & Poppendieck's principer och tankeverktyg vid namn LUT – Lean systemUTveckling.

Nyckelord: lean software development, systemutveckling, metod

Innehållsförteckning

1	Inledning	1
1.1	Bakgrund	1
1.2	Problemdiskussion	2
1.3	Forskningsfrågor	3
1.4	Syfte	3
1.5	Avgränsningar	4
1.6	InnovationLab	4
1.7	Målgrupp	4
1.8	Disposition	5
2	Vetenskapligt förhållningssätt och metod	6
2.1	Vetenskapligt perspektiv och förhållningssätt	6
2.1.1	Undersökningstyp	7
2.2	Forskningsstrategi	7
2.3	Metoder	8
2.3.1	Urvalsmetod	8
2.3.2	Insamlingsmetod	9
2.3.3	Analysmetod	11
2.3.4	Utvärderingsmetod	12
2.3.5	Presentationsmetod	13
3	Teori	14
3.1	Tidigare forskning	14
3.1.1	Systemutveckling och systemutvecklingsmetoder	14
3.1.2	Metodutveckling	14
3.1.3	Lean production och lean software development	14
3.2	Teoriområden kopplade till forskningsfrågorna	15
3.3	Lean	16
3.3.1	Olika former av <i>lean</i>	16
3.3.2	Lean production	18
3.3.3	14 principer för <i>lean production</i>	20
3.3.4	Lean software development	23
3.4	Metodbegreppet	24
3.4.1	Allmänt om metoder	24
3.4.2	Synsätt	25
3.4.3	Modell och modellstruktur	25
3.4.4	Verktyg	25
3.5	Agila metoder	26
3.6	Systemutveckling	26
3.7	Organisation	26
4	Empiri	27
4.1	Informella konversationer	27
4.2	Det abduktiva arbetssättet	30
5	Poppendieck's principer och tankeverktyg	31
5.1	Principer	31
5.2	Tankeverktyg	31
6	Resultat	33
6.1	Likheter/skillnader mellan Poppendieck's och Liker's principer	33
6.1.1	Basera beslut på långsiktig filosofi, inte kortsiktiga finansiella mål	33
6.1.2	Skapa ett kontinuerligt processflöde som lyfter problem till ytan	33
6.1.3	Använd pull-system för att undvika överproduktion	34

6.1.4	Jämna ut arbetsbelastningen	34
6.1.5	Skapa en kultur som stannar upp för att laga problem, gör det rätt från första början	34
6.1.6	Standardiserade uppgifter är grunden för ständigt lärande och bemyndigande av medarbetaren	34
6.1.7	Använd visuell kontroll för att inte låta problem vara dolda	34
6.1.8	Använd endast pålitlig, väl beprövad teknologi som stödjer medarbetare och processer	34
6.1.9	Skapa ledare som förstår arbetet, lever filosofin och lär ut den till andra	35
6.1.10	Skapa exceptionella människor och team som lever efter filosofin	35
6.1.11	Respektera nätverket av partners och leverantörer genom att utmana dem och hjälpa dem växa.....	35
6.1.12	Gå och se själv för att verkligen förstå situationen (Genchi genbutso).....	35
6.1.13	Ta beslut långsamt och med konsensus, betänk noga alla alternativ och implementera beslut snabbt	35
6.1.14	Bli en lärande organisation genom obehaglig reflektion (Hansei) och ständig förbättring (Kaizen)	36
6.1.15	Överensstämmelse mellan principerna	36
6.2	Problem och styrkor med Poppendieck's principer och tankeverktyg	37
6.2.1	Problem med beskrivningen av principer och tankeverktyg.....	38
6.2.2	Styrkor med Poppendieck's principer.....	40
6.2.3	Generella styrkor med den skapade metoden	46
6.2.4	Styrkor med Poppendieck's principer och tankeverktyg	47
6.3	Lean-metoden i relation till Goldkuhl's metodstruktur	49
6.4	Metoden LUT - Förslag till lösning på identifierade problem.....	50
6.4.1	Ledarskapets uppgifter	51
6.4.2	Förstudie	53
6.4.3	Möten som genomförs i systemutvecklingsprocessen	54
6.4.4	Synliggör information.....	59
6.5	LUT: Arbetsprocess.....	60
7	Slutsats.....	62
7.1	Slutsats	62
7.2	Utvärdering	63
7.2.1	Validitet i resultatet.....	63
7.2.2	Validitet i datainsamling	63
7.2.3	Validitet i dataanalys	63
7.2.4	Kommunikativ validitet	64
7.2.5	Generaliserbarhet	64
7.2.6	Kongruens	64
7.2.7	Relation till annan kunskap.....	64
7.2.8	Metodutvärdering.....	64
7.3	Förslag på fortsatt forskning	66
8	Litteraturförteckning	67
9	Bilagor	70
	Bilaga A - Ledarmall	70
	Bilaga B - Projektavslutsmall	71
	Bilaga C - Iterationsmall	72
	Bilaga D - Förstudiemall	75
	Bilaga E - Tisdag-torsdagmall	76
	Bilaga F - Kaizenmall	77
	Bilaga G - "6Månadersmall"	78
	Bilaga H - Diskussionsunderlag/resultat.....	79
	Bilaga I - Sammanfattning av Poppendieck's principer och tankeverktyg	88

Figurförteckning

Figur 1 - Former av intervjuer	9
Figur 2 - Teoriavsnitt relaterade till forskningsfrågorna.....	15
Figur 3 - Klargörande modell över begrepp	17
Figur 4 – Goldkuhl's (1991) metodstruktur.....	25
Figur 5 - Studiens abduktiva arbetssätt.....	30
Figur 6 - Likheter och skillnader mellan Poppendieck's och Liker's principer	36
Figur 7 - Problemgraf kring beskrivningen av Poppendieck's principer och tankeverktyg.....	38
Figur 8 - Styrkegraf avseende nöjdare kunder	40
Figur 9 - Problemgraf avseende ständig förbättring	40
Figur 10- Problemgraf avseende ledarskap.....	41
Figur 11- Problemgraf avseende synliggörande av information.....	42
Figur 12- Problemgraf avseende slöserier/värden	43
Figur 13- Problemgraf avseende utveckling.....	44
Figur 14 - Styrkegraf avseende defekter	45
Figur 15 - Metoden beskriven med Goldkuhl's (1991) modell.....	49
Figur 16 – LUT's arbetsprocess.....	60
Figur 17 - Kontinuerliga möten	61
Figur 18 - Aktiviteter utan fasta tidpunkter för utförande	61

Tabellförteckning

Tabell 1 - Liker's 4P och 14 principer	20
Tabell 2 - Skillnad mellan tillverkning och systemutveckling	23
Tabell 3 - Poppendieck's 7 principer för <i>lean software development</i>	31
Tabell 4 - Poppendieck's tankeverktyg	32

1 Inledning

1.1 Bakgrund

Systemutveckling präglas ofta av långa utvecklingsprocesser. Flertalet utvecklare har accepterat det som ett faktum att projekt inte möter uppsatta krav på tid och kostnad (Cook & Semouchchak, 2004). Problematik med buggar i mjukvaran och förskjutna deadlines uppstår kontinuerligt och leder till högre kostnader för företag och organisationer. För att förstå denna problematik bör mjukvara betraktas som mer än individuella rader kod. Det är en produkt med en hel livscykel där drift och förvaltning står för över 60 % av den totala kostnaden. Underhållskostnader som uppkommer till följd av systemutveckling kan undvikas genom att minska dessa brister. Om detta lyckas kan organisationer bättre hushålla med sina resurser. (Cook & Semouchchak, 2004)

För att komma till rätta med dessa problem behövs en systemutvecklingsprocess som belyser problem direkt när de uppstår och tvingar de inblandade till reflektion över processen som skapar problemen. Processen behöver belysa de bakomliggande orsakerna eftersom låg kvalitet på mjukvara är ofta ett symptom på djupare organisatoriska problem. (Middleton, 2001)

Långa utvecklingsprocesser blir även problematiska när vi vet att världen förändras snabbare och snabbare. Kraven på företag som arbetar med systemutveckling blir kontinuerligt större och större. Det som var aktuellt för en kund vid beställningstidpunkten kanske inte längre motsvarar det aktuella behov som kunden har efter ett halvår. Det är därför viktigt för systemutvecklingsorganisationer att snabbt kunna anpassa sig för att se till att kunden får det den vill ha, inte vad den har beställt. (Anderson, 2004)

Traditionella systemutvecklingsmetoder har ett fokus på tillverkaren (utvecklaren) och har utvecklats utifrån en lednings eller organisationssynvinkel (Cronholm 2008). Som en reaktion på de traditionella metoderna har de Agila metoderna växt fram där individer och interaktion, fungerande mjukvara, samarbete med kunden och en förmåga att reagera på förändring är de grundläggande värderingarna (Anderson, 2004). Agila metoder fokuserar därmed både på tillverkaren och på kunden (Cronholm, 2008) vilket jag ser som en förutsättning för att möta de ovan beskrivna problemen.

Ett synsätt som syftar till att låta organisationer möta dessa krav är *lean software development* (Poppendieck & Poppendieck, 2003). Det baserar sig på tankar och principer inom *lean production* vilket är en produktionsteknik Toyota skapat och använt för att arbeta effektivt och hålla hög kvalitet på sina produkter under flera decennier men som först nyligen har kommit att appliceras på systemutveckling. (Poppendieck & Poppendieck, 2003)

Det är dock svårt att direkt applicera det tankeverktyg som Poppendieck & Poppendieck's (2003) presenterar i sin bok *Lean software development – an agile toolkit* i en verksamhet. För att använda dessa verktyg behövs en konkret metod (se avsnitt 6.2).

1.2 Problemdiskussion

Det finns ett evigt växande utbud av filosofier och metoder för hur effektiv och högkvalitativ systemutveckling ska ske. Ett av de senare tillskotten till denna flora av metoder är *lean software development* (se 3.3.4).

De flesta systemutvecklingsorganisationer arbetar idag med ett flertal parallella projekt som kan genomföras med hjälp av olika agila metoder¹ (se avsnitt 3.5). Detta kan dock uppfattas som ett problem om en organisation inte har ett standardiserat arbetssätt då utvecklare själva väljer på vilket sätt varje enskilt projekt ska genomföras. När det finns en sådan fragmentering i arbetet söker företag att standardisera sitt arbete med hjälp av metoder för att få konkurrensfördelar och för att höja kvaliteten på sitt arbete. (Cronholm, 1995)

Poppendieck & Poppendieck's (2003) beskriver sju principer för hur man kan arbeta med *lean software development*. Dessa principer har 22 tankeverktyg kopplade till sig och följs åt av ett antal förslag till handling som beskriver vad som ska göras i organisationen. Dessa saknar enligt mig en koppling till hur en systemutvecklingsorganisation konkret kan använda dessa handlingar, hur dessa handlingar ska dokumenteras för att möjliggöra ett kontinuerligt lärande och hur organisationens arbetsprocess skall se ut (se avsnitt 6.4). Det finns med andra ord problem för användare att kunna tillgodogöra sig en tillräcklig förståelse för att kunna använda principerna, verktygen och handlingarna på ett tillfredställande sätt. För att möta dessa problem behövs det en metod vilken konkret visar hur de kan användas inom en systemutvecklingsorganisation.

En del av inspirationen bakom denna studie väcktes när jag läste en kandidatuppsats där Karlsson (2008) beskriver Poppendieck & Poppendieck's (2003) sju principer som en vald kvalitetsmodell för systemutvecklingsprocessen. Teori har där jämförts gentemot ett It-företags arbetsmetoder för att därefter genomföra en jämförande studie i syfte att se om och i så fall hur *lean software development* skulle kunna implementeras i verksamheten med hjälp av Scrum (en agil systemutvecklingsmetod) samt vilka effekter detta hade fått på kvalitetsarbetet (Karlsson, 2008). Rekommendationen att använda Scrum styrker mitt påstående om att det finns problem med beskrivning av principerna och tankeverktygen.

Det finns mer skrivet om metoder för agil utveckling (t.ex. Scrum) än metoder för *lean software development*. Vissa är normativa men det finns oftast en substantiell skillnad mellan böckers version av metoder och hur den faktiskt används i praktiken (Conboy & Duarte, 2010). Detta leder mig till att se att metoden bör tas fram i samarbete med praktiker för att undvika denna diskrepans.

Praktiker har även ett uttalat behov av att förstå hur *lean software development* kan användas i en systemutvecklingsorganisations dagliga arbete.² Att det som finns skrivet inom området saknar klart definierade handlingar som ska genomföras utan att ha dessa grupperade i aktiviteter, händelser eller processer uppfattas som ett problem. Det finns således ett behov av att göra förbättringar av Poppendieck & Poppendieck's (2003) principer och tankeverktyg för att de skall kunna användas som ett underlag för att skapa en metod.

¹ Konversation med föreståndare för InnovationLab 2011-03-17

² Konversation med föreståndare för InnovationLab 2011-03-17

1.3 Forskningsfrågor

Studien vill genom att skapa en metod för *lean software development* baserad på Poppendieck & Poppendieck's (2003) principer och tankeverktyg besvara frågan hur dessa principer och tankeverktyg konkret kan användas i praktiken. För att göra detta behöver jag förstå vilka allmänna problem *lean production* och *lean software development* ämnar lösa samt vilka dess styrkor är. Dessutom behöver jag förstå vilka problem och styrkor som finns avseende Poppendieck & Poppendieck's (2003) principer och tankeverktyg i synnerhet.

Detta leder mig fram till studiens forskningsfrågor:

1. Hur kan en konkret metod för *lean software development* baserad på Poppendieck & Poppendieck's (2003) principer och tankeverktyg se ut?
 - 1.1. Vilka styrkor och problem beskriver *lean production* och *lean software development* i allmänhet?
 - 1.2. Vilka styrkor och problem finns avseende beskrivning av Poppendieck & Poppendieck's (2003) principer och tankeverktyg i synnerhet.

1.4 Syfte

Studiens syfte är att skapa en metod som möjliggör för organisationer att använda Poppendieck & Poppendieck's (2003) tankeverktyg för *lean software development* i praktiken.

I processen att ta fram underlag för denna metod skapar studien insikter om de övergripande problem som *lean production* och *lean software development* syftar till att lösa.

Studien syftar även till att göra en analys av de styrkor och problem som finns gällande beskrivning av Poppendieck & Poppendieck's (2003) principer och tankeverktyg specifikt.

1.5 Avgränsningar

Endast en organisation har studerats i studien för att det enligt mig har varit viktigt att gå in på djupet i frågeställningen. Detta medför en minskad generaliserbarhet men ger ett gott underlag för vidare forskning där resultatet vidare kan prövas mot andra organisationer. Jag har därför valt att endast använda mig av en kvalitativ källa men att djupgående studera denna för att få en så bra bild som möjligt av underlaget.

Det finns ett stort antal agila metoder som möjligtvis skulle kunna användas för att besvara den huvudsakliga forskningsfrågan på samma sätt som Karlsson (2008) har gjort. Denna studie väljer dock att begränsa sig till att använda Poppendieck & Poppendieck's (2003) principer och tankeverktyg som underlag för att skapa en metod eftersom de enligt mig är väletablerade inom området, kommunicerbara, bekanta för praktiker och tillgängliga.

Studien syftar till att föreslå en metod och inte att implementera eller att studera effekter av användning. Att studera implementationen och användandet av metoden i verksamheten kommer inte att vara möjligt inom studiens tidsram.

Systemutvecklingsprocessen studeras på en övergripande nivå där detaljer som att välja utvecklingsmiljöer och konkret problemlösning är uppgifter som ligger på utvecklarteam och därför inte behandlas i detalj i denna studie. Studien behandlar även endast en begränsad del av aktiviteterna kring utformning av avtal, projektacceptans samt drift och förvaltning av systemutvecklingsprojekt.

1.6 InnovationLab

I studien har systemutvecklingsorganisationen InnovationLab studerats. Organisationen är en del av Högskolan i Borås som strävar efter att fånga upp den praktik som sker utanför Högskolan inom området informatik och systemutveckling. Högskolan i Borås som helhet har ett kompetensmässigt överskott när det gäller metoder och modeller inom design av IT-användning. Ett mål med InnovationLab är att tillsammans med andra aktörer skapa förutsättning att etablera en någorlunda stark kompetens inom realisering av IT-användning. (Lind et al., 2006) Organisationen arbetar mot detta mål genom att bedriva systemutvecklingsprojekt för interna och externa kunder³.

1.7 Målgrupp

Studien kommer förhoppningsvis att vara användbar för praktiker som vill använda *lean software development* i en systemutvecklingsorganisation. Även verksamhetsledare i näringslivet som önskar öka sin kunskap om hur *lean production* kan användas i andra domäner än inom den traditionella tillverkningen kan ha nytta av denna studie.

Akademiker som är intresserade av vilket underlag som behövs för att påbörja en implementering av systemutvecklingsmetoder kommer förhoppningsvis att ha glädje av studiens förslag till vidare forskning. Det är även tänkbart att analysen av principerna och tankeverktygen kan vara användbara för akademiker som är intresserade av hur förbättringsförslag av principer och tankeverktyg kan inhämtas.

³ Konversation med föreståndare för InnovationLab 2011-03-17

Studien är även intressant för människor som har ett allmänt intresse av metoder och systemutveckling.

1.8 Disposition

Detta avsnitt beskriver hur rapporten är strukturerad och syftar till att ge läsaren en förståelse för studiens olika delar.

Kapitel 1 – Inledning

Det inledande kapitel ger en introduktion till studiens bakgrund genom att förklara vilket problem som studeras och vilka de bakomliggande orsakerna till detta är. Här presenteras även studiens huvudsakliga forskningsfråga samt delfrågor.

Kapitel 2 – Vetenskapligt förhållningssätt och metod

Genomgång och motivering av det vetenskapliga förhållningssätt och metod som används för att besvara studiens forskningsfrågor. Kapitlet beskriver även hur datainsamling, analys och den övergripande forskningsprocessen ser ut.

Kapitel 3- Teori

Genomgång av de koncept och den teori som ligger till grund för studien. Läsaren ska efter detta kapitel förstå bakgrunden till de begrepp och resonemang som används i resterande del av studien.

Kapitel 4 – Beskrivning av Poppendieck & Poppendieck's principer och verktyg

I detta kapitel görs en kort beskrivning av Poppendieck & Poppendieck's (2003) principer och tankeverktyg för *lean software development*. Dessa är vad som i detalj har behandlats i studien och vad som resultatet grundas på.

Kapitel 5 – Resultat

Besvarar forskningsfrågorna genom att först besvara delfrågorna för att sist besvara huvudfrågan. Delfrågorna besvaras genom att först göra en jämförelse av likheter och skillnader mellan *lean software development* och *lean production*. Graden av överensstämmelse visar huruvida de möter samma problem och innehar samma styrkor. För att identifiera dessa och för att besvara den andra delfrågan görs en problemanalys av Poppendieck & Poppendieck's (2003) principer och tankeverktyg varpå en styrkeanalys genomförs.

Kapitel 6 – Slutsats

Presenterar det kunskapsbidrag som studien tillför och en reflektion över resultatet görs. Här presenteras även förslag till fortsatt forskning.

Kapitel 7 – Litteraturförteckning

Presenterar de källor som har använts i studien.

Kapitel 8 – Bilagor

I kapitlet återfinns de mallar som kan användas som en del i metoden LUT, en djupare sammanfattning av Poppendieck & Poppendieck's (2003) principer och tankeverktyg samt underlag från diskussioner med praktiker.

2 Vetenskapligt förhållningssätt och metod

2.1 Vetenskapligt perspektiv och förhållningssätt

Studien inbegriper till stora delar teoribildning vilket kräver att jag intar en öppen och engagerad forskarroll där jag betraktar min förkunskap som en tillgång i studien och tillåter att den färgar resultatet. Genom att låta detta ske så kan jag inte inta en helt objektiv forskarroll men med tanke på att konversationer tillsammans med praktiker är det som ska leda fram till resultatet betraktar jag detta som ofrånkomligt, det skulle vara kontraproduktivt att försöka anlägga en objektiv forskarroll i studien då mycket av den teori som skapas endast kan uppstå i dessa konversationer. (Patel & Davidson, 2003)

Praktiken är långt ifrån den kontrollerade laboratoriemiljö som positivister föredrar för att under kontrollerade former verifiera hypoteser (Oates, 2006). För att genomföra studien enligt detta har jag valt att anlägga ett hermeneutiskt perspektiv på forskningen vilket innebär att jag genom att tolka information utifrån min egen förförståelse skapar kunskap om studieobjektet (Patel & Davidson, 2003).

Hermeneutiken vänder sig emot positivismens påstående om att det finns en absolut sanning. Genom att tolka och analysera språket genom samtal och texter skapar jag mig en rikare bild där helheten i tolkningen är viktigare än ett mätbart slutresultat som antingen kan bevisas eller motbevisas. Resultatet i studien kommer att uppstå genom holism vilket innebär att summan av delarna är vad som utgör helheten. Det är först när delarna sätts samman som resultatet kan uttolkas. (Patel & Davidson, 2003)

Detta medför ett antal effekter som kan uppfattas som negativa för forskningsprocessen. Dels att min egen förförståelse kan påverka studiens resultat och att den får styra vad som studerats (Patel & Davidson, 2003). Jag kan inte frigöra mig från det historiska och sociokulturella arvet, det medför att jag utgår utifrån egna värderingar och fördomar vilket även påverkar den kunskap som inhämtas eftersom jag inte kan frigöra mig från dessa (Egidius, 1986).

Inom forskning skiljs det ofta på kvantitativt och kvalitativt arbete. (Andersen, 1994b) Kvalitativt arbete är i normalfallet förknippat med hermeneutiskt och tolkande arbete medan kvantitativt förknippas med positivism där resultat kan mätas numeriskt. (Patel & Davidson, 2003). Kvantitativa forskare har som utgångspunkt att det som studeras kan göras mätbart medan kvalitativa forskare menar att varje fenomen innehåller en unik kombination av egenskaper och kvaliteter som inte kan mätas. (Andersen, 1994b)

Studien är genomgående av kvalitativ karaktär. Det som studeras är inte lämpligt att mäta eftersom det genomgående förutsätts att ny teori ska skapas och valideras tillsammans med praktiker.

Början och slutet på tolkningen är svår att förutse då jag enligt den hermeneutiska spiralen kontinuerligt utvecklar tolkning, förståelse och produktion av ny text (Patel & Davidson, 2003). Det som var avgörande för hur länge spiralen kunde fortgå är studiens tidsaspekter.

2.1.1 Undersökningstyp

Studiens syfte utgör en grund för att beskriva vilken typ av undersökning som har genomförts. Olika typer av undersökningar används i olika faser av studien. Den inledande litteraturstudien är en beskrivning kring hur det är i dagsläget vilket resulterar i en genomgång av forskning inom fältet. Litteraturstudiens resultat används som underlag för att identifiera problem och styrkor med *lean software development*, *lean production* och Poppendieck & Poppendieck's (2003) principer och tankeverktyg men även som komplement i processen att skapa en metod för *lean software development*.

Efter det anläggs ett explorativt arbete vid metodskapandet för att beskriva hur en organisation bör arbeta (Patel & Davidson, 2003) och vilka problem som finns beskrivning av principerna och tankeverktygen. Detta görs genom att underlag införskaffat under litteraturstudier djupgående diskuteras och valideras med praktiker. I denna process identifieras ytterligare problem gällande beskrivning av Poppendieck & Poppendieck's (2003) principer och tankeverktyg. De insikter som skapas i studien tillkommer genom explorativa studier där det beskrivs hur det kan vara snarare än hur det faktiskt är (Patel & Davidson, 2003).

Genom att studien utvecklar en metod för arbete med *lean software development* kommer även resultat att vara av en metodutvecklande typ. (Goldkuhl, 1998)

2.2 Forskningsstrategi

För att besvara den huvudsakliga forskningsfrågan har ett abduktivt arbetssätt anlagts där Poppendieck & Poppendieck's (2003) principer tankeverktyg hämtade från teori utgör underlag för att tillsammans med praktiker skapa en metod för *lean software development*.

Teori ifrån litteraturstudier studeras deduktivt för att skapa ett underlag att diskutera tillsammans med föreståndare för InnovationLab. Efter att detta underlag har diskuterats och synpunkter inhämtats så skapas ett nytt teoretiskt underlag. Detta underlag har syftat till att validera de tankar som i teorin har presenterats som lämpliga (se bilaga H). Underlaget används induktivt för att generera ny teori i form av en metod för *lean software development*.

Arbetsättet ligger nära en beskrivning av den hermeneutiska spiralen som innebär att man utgår från förförståelsen och vidareutvecklar den för varje studie. (Patel & Davidson, 2003) Det innebär inte att det är synonymt med abduktion, men jag väljer att använda liknelsen för att motivera varför det är ett abduktivt angreppssätt som anläggs på studien.

2.3 Metoder

2.3.1 Urvalsmetod

Poppendieck & Poppendieck's (2003) 22 tankeverktyg för *lean software development* används i studien som underlag för metodskapandet. Jag har valt att arbeta med dessa tankeverktyg eftersom de är väletablerade inom området, kommunicerbara, tillgängliga samt eftersom det finns ett intresse hos praktiker att se hur dessa kan användas i praktiken.

InnovationLab har främst valt för att det är en systemutvecklingsorganisation som har uttryckt en önskan om att använda en metod för att arbeta med *lean software development*⁴. Tillgängligheten till organisationens föreståndare är dessutom väldigt god vilket gör den lämplig för mig att studera i denna studie då studien syftar till att göra en djupgående analys. Det bör dock påpekas att eftersom implementeringen av metoden inte studeras så är det endast föreståndarens uppfattning av situationen som har kunnat användas som underlag, inte effekterna av användning. Det är rimligt att anta att uppfattning av situationen kan skilja sig från hur den kommer att uppfattas i verkligheten vid användning. Önskvärt skulle ha varit att kunna studera den färdiga metoden vid användning för att se hur den stämmer överens med verkligheten.

För att göra ett urval till litteraturstudien har jag främst utgått från de artiklar och böcker som har haft flest citeringar inom ämnena *lean production* och *lean software development* samt genom att diskutera lämpliga källor med föreståndaren för InnovationLab. Vid egna sökningar har jag sett att Womack et al (1990) och Liker (2004) är frekvent citerade inom området *lean production*. Det framkommer även att Womack et al. (1990) är de som har gjort några av de tidigaste studierna kring ämnet och även de som gjort den mest övergripande studien sett utifrån ett geografiskt och tidsmässigt perspektiv.

Inom området *lean software development* framkom det genom sökningar på ”lean software development” och närliggande begrepp såsom exempelvis ”lean development” att Poppendieck & Poppendieck's (2003) var de överlägset mest citerade inom området. För att ytterligare visa på deras relevans bör det nämnas att de även citeras ofta i artiklar som inte direkt behandlar *lean software development* utan även citeras i övergripande artiklar kring agila metoder.

Vad det gäller ytterligare källor är utbudet av artiklar och böcker skrivna av akademiker relativt begränsat och mycket av det som har skrivits är då i form av referat eller fallstudier. Jag har då gjort egna bedömningar av trovärdigheten och relevansen samt satt dessa i relation till vilka påståenden de stödjer eller bestrider i min text och då noga sett till att lägga lika stor vikt vid positiva och negativa källor. Detta har även medfört att jag har ansett det viktigt att göra en djupgående studie med stora mängder datainsamling för att inte helt förlita mig på tidigare studier. Resultatet av detta ställningstagande blev att jag valde att förlita mig på endast en kvalitativ källa i form av InnovationLab's föreståndare då tillgången var god och flera längre möten med denne kunde genomföras löpande under studien.

Jag anser även att det var viktigt att metoden skapades tillsammans med praktiker. Detta eftersom metoder ofta baseras på tidigare framgångsfaktorer och inte är statiska artefakter utan något som behöver leva vidare och utvecklas efter att de har skapats (Cronholm, 2008).

⁴ Konversation med föreståndare för InnovationLab 2011-03-17

Att de som ska använda metoden har en förståelse för hur den har uppkommit blir därför viktigt. Ytterligare ett argument för att göra detta tillsammans med praktiker är principen *kaizen* (ständig förbättring) som finns inom *lean production*. Studien kopplar samman metodskapande och dessa tankar vilket jag anser som ett relevant argument för att studien genomförs i samarbete med praktiker för att redan från början föra in tankarna om hur *lean software development* ska användas. D.v.s. i nära kontakt med kunden och med en strävan mot kontinuerlig förbättring av arbetet.

Genom att jag involverar praktiker i framtagandet av denna metod skiljer sig resultatet i denna studie och Karlssons (2008) studie avsevärt. Båda diskuterar till viss grad hur *lean software development* kan användas för att höja kvaliteten men med olika förslag till lösningar och olika angreppssätt. En stor del av skillnaden har kunnat skapas genom att jag valt att göra en så pass djupgående studie tillsammans med föreståndaren för en enskild organisation.

2.3.2 Insamlingsmetod

Jag har samlat in data genom litteraturstudier och informella konversationer med föreståndaren för InnovationLab. Litteraturstudier är det som har tagit längst tid i studien eftersom varje informell konversation har identifierat luckor i mitt eget vetande som är relevanta för studien. Det är rimligt att anta att varje timme av informella konversationer har resulterat i 3-5 timmar ytterligare litteraturstudier.

Informella konversationer är en form av intervju med en låg grad av standardisering. Patton (2002) beskriver tre huvudsakliga former av intervjuer: Informell konversation, allmän intervjuguide och standardiserade intervjuer.



Figur 1 - Former av intervjuer

Som figur 1 beskriver har dessa olika intervjutyper olika grader av standardisering där informell konversation är den med lägst grad av standardisering. Detta eftersom den förlitar sig på spontanitet och har en stor flexibilitet kring vad som diskuteras. Svaren som respondenten ger styr resterande del av samtalet. Detta medför ett antal styrkor eftersom intervjuformen är väldigt flexibel och uppmuntrar spontanitet. Svagheter med intervjuformen är att det krävs ett omfattande och tidskrävande arbete för att samla in information vilket ställer stora krav på att den som genomför intervjun på ett naturligt sätt kan styra konversationen. (Patton, 2002)

Den allmänna intervjuguiden är precis som figur 1 visar ett mellanting mellan den informella konversationen och den standardiserade intervjun. Skillnaden mot den informella konversationen är att man utgår ifrån ett antal förberedda frågeställningar av en allmän karaktär. Det går i denna form av intervjuer att ändra på när frågorna ställs i tiden och att

ställa följdfrågor. Genomför man däremot en standardiserad intervju så skall samma frågor ställas i samma följd vid varje intervjutillfälle. (Patton, 2002)

För att få en förståelse för hur organisationen arbetar idag genomfördes en affärsprocessmodellering tillsammans med föreståndaren för den studerade organisationen. Detta gav oss en samsyn kring hur verksamheten fungerar i dagsläget och utgjorde ett bra underlag att falla tillbaka på vid de informella konversationerna när det uppkom oklarhet i hur det vi diskuterade relaterade till det sätt som organisationen fungerar idag.

Det teoretiska underlaget som användes vid resterande informella konversationer genererades med hjälp av litteraturstudier. Detta eftersom studien till stor del behandlar ett redan existerande fenomen, *lean production*, som appliceras i systemutvecklingsdomänen. För att kunna bilda mig en uppfattning om hur denna applicering på systemutveckling skiljer sig mot den traditionella tillverkningsdomänen behövde jag se dels aktuell forskning inom området men även den historiska bakgrunden. Litteraturstudien ger mig även en initial uppfattning kring de styrkor och svagheter som finns gällande *lean software development* genom Poppendieck & Poppendieck's (2003) principer och tankeverktyg.

Utifrån detta material kunde jag härleda ett antal teman att diskutera med föreståndaren. Bland annat hur Poppendieck & Poppendieck's (2003) principer och tankeverktyg kan användas i praktiken samt deras lämplighet för den studerade organisationen. Underlaget kompletterades efter varje konversation med nytt material som tillgängliggjordes för föreståndare i förväg och därför kunde fungera genomgående som ett underlag vid framtida konversationer. Inför varje konversation enades vi kring vilka teman som skulle behandlas och använde därefter fritt underlaget för att föra konversationen framåt.

På dessa möten skapades dokumentation under tiden Poppendieck & Poppendieck's principer och tankeverktyg diskuterades. Denna försökte vi att hålla så kort som möjligt utan att tappa viktig information. Med hjälp av detta blev det enklare att styra konversationen eftersom föreståndare fick en möjlighet att formulera sig på papper direkt under konversationen vilket identifierade eventuella luckor i resonemang och bidrog till ett jämnt flöde i konversationen.

I övrigt spelades inte intervjuer in eller dokumenterades eftersom det hade brutit flödet i processen. Det var viktigt för mig att dessa konversationer kunde ske helt obehindrat och att föreståndaren kunde känna sig fri att tala om vad som helst på vilket sätt som helst utan att oroa sig för att detta lagrades och offentliggjordes. Om denna oro hade funnits finns det en risk att det naturliga informella samtalet som sker mellan två parter som hyser tillit till varandra inte hade uppstått. Att inte bryta flödet i konversationen är även viktigt enligt Patton (2002). Patton (2002) beskriver även att det under informella konversationer är vanligt att i efterhand göra anteckningar kring konversationen istället för att dokumentera eller spela in dessa.

Efter detta skapades ett förslag till underlag för en metod som illustrerade hur verktygen praktiskt skulle användas i verksamheten genom att samla alla förslag till vad som ska göras i en arbetsprocess. Denna nya arbetsprocess byggde på den arbetsprocessmodellering som genomfördes i studiens början⁵. Efter att denna diskuterats med föreståndaren tillkom

⁵ Modellering av nuvarande arbetsprocess genomfördes under 2 möten 2011-03-18 och 2011-03-28. Denna har inte använts som underlag vid skapandet av metoden annat än som referensmaterial för att se vilka aktiviteter organisationen i dagsläget behandlar.

kompletterande steg som behövdes för att metoden skulle kunna fungera som en helhet vilket gjorde att förslaget till underlag uppdaterades och diskuterades ännu en gång. Resultatet från dessa diskussioner återfinns i studiens resultatkapitel (se kapitel 6).

Totalt genomfördes åtta informella konversationer vilka pågick mellan två och tre timmar. En sammanfattande beskrivning från dessa konversationer återfinns i bilaga H där temat för varje möte också finns beskrivet.

Validiteten i det insamlade materialet har säkrats genom att tillsammans med föreståndaren gå igenom och matcha teorin mot den verklighet som praktiker befinner sig i. Detta skapade förståelse för teorins lämplighet och på vilket sätt metoden kunde skapas utifrån teorin för att passa verkligheten och inte endast de teoretiska scenarier som litteraturen beskrev.

2.3.3 Analysmetod

Problem och styrkor med *lean software development*, *lean production* och Poppendieck & Poppendieck's principer och tankeverktyg har identifierats genom att genomföra en problemanalys och en styrkeanalys.

Syftet med att genomföra dessa analyser är att hitta olika typer av förändringsåtgärder genom att ställa en diagnos på problem (Goldkuhl & Röstlinger, 1988).

Det innebär att jag förutsättningslöst ifrågasätter det som förefaller vara oproblematiskt och går från vaga problemuppfattningar till preciserade beskrivningar (Goldkuhl & Röstlinger, 1988). En viktig del i förändringsanalysarbetet är att etablera avgränsningar vilket jag också har gjort i kapitel 1.2.4.

Med problem menar Goldkuhl & Röstlinger (1988) skillnaden mellan "hur jag vill att det ska vara" och "hur jag uppfattar att det är".

För att hitta problem kring *lean software development* och *lean production* generellt fick jag först klarlägga vilka likheter och skillnader som finns dem emellan. Detta för att se om separata enskilda analyser behövde genomföras. Detta gjordes genom att se hur väl det Liker's (2004) principer för *lean production* behandlar stämmer överens med vad Poppendieck & Poppendieck's (2003) principer för *lean software development* behandlar. När detta hade klarlagts kunde jag se att de i all väsentlighet behandlar samma sak fast inom olika domäner. Detta gjorde att jag kunde beskriva styrkor och problem med dem båda genom att genomföra en problem- och styrkeanalys av Poppendieck & Poppendieck's (2003) principer och tankeverktyg för *lean software development*.

Samtidigt som kartläggningen av styrkorna gjordes så genomfördes även de informella konversationerna tillsammans med föreståndaren för den studerade organisationen. Under tiden för dessa skapade jag ett utkast som identifierade de generella problemen som finns kring beskrivning av principerna och tankeverktygen. Underlaget växte därefter fram iterativt under konversationer och kompletterades löpande med tankar som har uppenbarats sig under litteraturstudien.

2.3.4 Utvärderingsmetod

Kvalitet i kvalitativa studier är något som är viktigt och som i fallet för kvalitativa studier omfattar hela forskningsprocessen. (Patel & Davidson, 2003)

De utvärderingskriterier som jag har valt att utvärdera studien efter är:

- Validitet i datainsamling
- Validitet i dataanalys
- Validitet i resultatet
- Kommunikativ validitet
- Generaliserbarhet
- Kongruens
- Relation till annan kunskap

Genom att utvärdera studiens validitet vill jag säkra att rätt företeelse har studerats. Detta går att stärka med hjälp av exempelvis en god teoriunderbyggnad. (Patel & Davidson, 2003)

Studien strävar efter att upptäcka företeelser vid den studerade organisationen samt att tolka och förstå hur systemutveckling kan bedrivas med hjälp av en metod för *lean software development*. Detta ligger i linje med Patel & Davidsons (2003) beskrivning av validitet inom kvalitativa studier. De säger att ambitionen är att upptäcka företeelser, tolka och förstå innebörden av livsvärlden samt att beskriva uppfattningar eller en kultur. Validitet inom kvalitativa studier syftar till att validera hela forskningsprocessen.

Inom kvantitativ forskning talar Patel & Davidson (2003) om reliabilitet som ett bra utvärderingsinstrument för studier där samma resultat skall uppkomma vid olika tillfällen. Inom kvalitativa studier går det dock att betrakta olika svar vid olika tillfällen som en tillgång då exempelvis intervjuade människor kan ändra uppfattning från beroende på om de har lärt sig något nytt eller fått tillfälle att reflektera över den tidigare intervju-sessionen.

Begreppen validitet och reliabilitet är inom kvalitativ forskning ofta så sammanflätade att de i stort sett blir utbytbara (Patel & Davidson, 2003), jag väljer därför att endast tala om validitet och ej reliabilitet vid utvärderingen.

Validitet är inte heller kopplat enbart till datainsamling utan syftar till att utvärdera validitet i alla forskningsprocessen olika delar som en helhet. När det gäller datainsamling så skall validiteten visa på huruvida forskaren har lyckats skaffa ett så pass komplett underlag som möjliggör en tolkning av dennes världsbild samt om det går att fånga det mångtydiga och kanske motsägelsefulla. (Patel & Davidson, 2003)

Analys av datainsamling görs genom att studera huruvida de olika inre logiska delarna går att relatera till en meningsfull helhet. Detta utvärderas genom att avgöra huruvida läsaren kan bygga sig en egen uppfattning kring de tolkningar som har gjorts. Patel & Davidsson (2003) kallar detta för den kommunikativa validiteten.

Studien har till stor del studerat hur en metod för *lean software development* kan se ut för en systemutvecklingsorganisation. Det är därför lämpligt att studera studiens resultat genom att studera hur metodens beståndsdelar har validerats hos praktiker. Studien utvärderas efter hur den kvalitativa analysen har lett fram till en förståelse av ett fenomen och vilka variationer

som detta fenomen uppvisar till sin kontext i enlighet med Patel & Davidssons (2003) beskrivning av hur generaliserbarhet kan utvärderas.

Att analysera kongruensen visar huruvida studien följer en röd tråd och utgör en sammanhängande helhet. Utvärderingen skall visa huruvida läsaren kan uppfatta att de olika delarna i uppsatsen har vävts samman till en sammanhållen argumentation. (Högskolan i Borås, IDA, Informatik 2003)

Genom att utvärdera hur studien förhåller sig till tidigare kunskap blir det enklare för läsaren att ta ställning till huruvida ny kunskap har skapats i studien baserat på vad som sedan tidigare har studerats inom området. (Högskolan i Borås, IDA, Informatik 2003)

2.3.5 Presentationsmetod

Studien presenteras på ett sätt som gör att såväl akademiker som praktiker kan tillgodogöra sig de delar av studien som är relevanta för dem. Detta görs genom att tydligt bryta ut studiens olika delar i olika kapitel som kan läsas var för sig och genom hänvisningar till andra kapitel påvisa vart det går att införskaffa en tydligare bild av det som beskrivs.

För att göra studien mer begriplig har jag valt att använda mig av modeller och tabeller där lämpligt. Exempelvis fokuserar resultatet till större del på modeller och tabeller eftersom de möjliggör att snabbare tillgodogöra sig den mest relevanta informationen medan teori och andra kapitel som i första hand är intressanta för akademiker fokuserar till större grad på textuella beskrivningar.

Genom att studien har en stor volym av bilagor har innehållet kunnat kondenseras och valet att fördjupa sig har lämnats till läsaren genom att göra referenser till relevanta bilagor där lämpligt.

3 Teori

3.1 Tidigare forskning

Avsnittet syftar till att lyfta fram tidigare relevant forskning inom de områden som studien behandlar. Det beskriver även vilken kunskap som ligger till grund för den kunskap som skapas inom denna studie.

3.1.1 Systemutveckling och systemutvecklingsmetoder

Andersen (1994a) beskriver i sin bok ”Systemutveckling – principer metoder och tekniker” den forskning som han har utträttat kring ämnet systemutveckling. Boken är relevant för studien då den ger en bra heltäckande bild kring ämnet systemutveckling.

Stefan Cronholm beskriver i sin rapport ”Using Agile Methods ? – expected effects” hur systemutveckling har gått ifrån traditionella till agila systemutvecklingsmetoder. Denna rapport är av stor nytta för mig då den beskriver både fördelar och nackdelar med agila metoder. Cronholm har även författat ett stort antal övriga rapporter kring ämnet.

Beskrivningen som Johansson et al. (2009) gör av agila metoder i sin kandidatuppsats har varit av nytta för studien då den påvisar hur väl agila metoder fungerar i praktiken.

3.1.2 Metodutveckling

Göran Goldkuhl har sedan 80-talet studerat metodutveckling och är väletablerad inom området bland annat genom de publikationer som har getts ut av forskningsgruppen VITS. Speciellt relevanta för min forskning har publikationerna ”Stöd och struktur i systemutvecklingsprocessen” och ”Metodanalys” varit medan en stor del andra publikationer har fungerat som inspiration snarare än regelrätt källa.

3.1.3 Lean production och lean software development

Womack et al. (1990) beskriver i sin bok ”The machine that changed the world” hur The Toyota Production system fungerar genom djupgående studier av Toyota och andra företag som använder sig av Lean. Denna bok är i högsta grad relevant för min forskning.

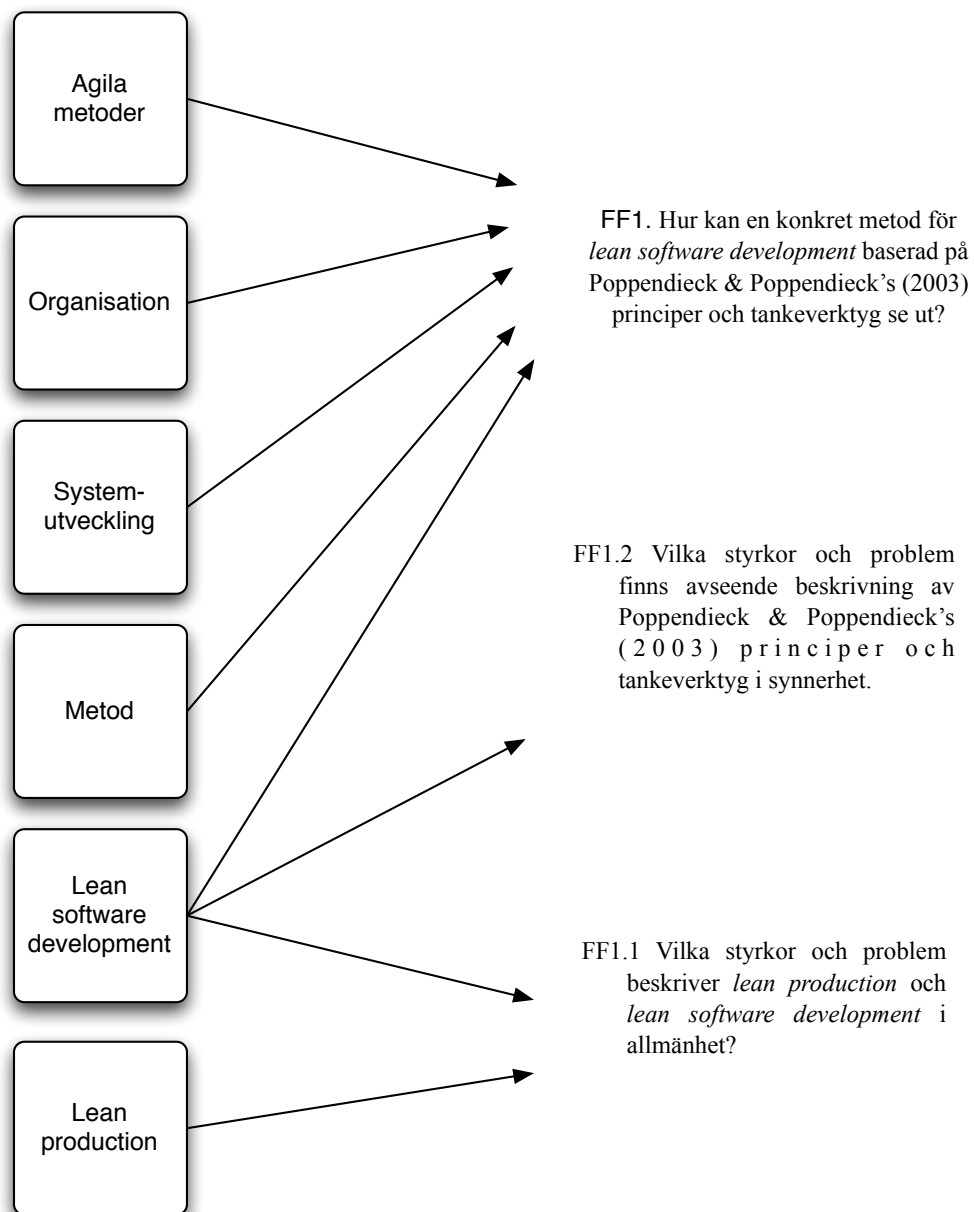
Liker beskriver i sin bok ”The Toyota Way” 14 principer för *lean production* som är intressanta för denna studie som ett verktyg för validering av principer inom lean software development.

Mycket av det som har skrivits inom området *lean software development* har skrivits av praktiker. De praktiker som har skrivit mest kring området är Poppendieck & Poppendieck (2003) och de är även de som har citerats av flest akademiker. Bidraget som akademien har gjort kring ämnet är relativt tunt och mycket är i form av fallstudier.

Denna studie har inspirerats av Karlssons (2008) kandidatuppsats kring *lean software development*, denna uppsats behandlar dock inte ämnet utförligt utan använder Scrum för att uppfylla Poppendieck & Poppendieck’s (2003) principer.

3.2 Teoriområden kopplade till forskningsfrågorna

Grafen beskriver hur de olika teoriområden som presenteras i kapitlet används för att besvara forskningsfrågorna.



Figur 2 - Teoriavsnitt relaterade till forskningsfrågorna

3.3 Lean

Syftet med att beskriva *lean production* ur ett historiskt och principiellt perspektiv är att jag vill ge läsaren den grund som behövs för att förstå hur studien förhåller sig till *lean software development* och *lean production*. Jag ger här även läsaren en introduktion till området som underlag för att förstå styrkor och problem som diskuteras i studien.

För att göra detta görs först en genomgång kring hur olika begrepp som behandlar *lean* förhåller sig till varandra. Därefter beskrivs övergripande *lean production* och *lean software development*.

Eftersom studien syftar till att skapa en metod för *lean software development* görs därefter en genomgång kring vad en metod är och vad den bör bestå av. I detta avsnitt beskrivs även en metodstruktur som jag senare prövar den fullständiga metoden mot för att visa på att resultatet är teoretiskt förankrat.

Kapitlet avslutas genom att introducera andra begrepp som kan vara viktiga att förstå för att tillgodogöra sig studien på bästa sätt.

3.3.1 Olika former av *lean*

När jag sökte på ”*lean development*” på Google scholar fick jag 873 träffar. Antal träffar för ”*lean software development*” var 911. Det första begreppet visade sig till stor del vara synonymt med det andra begreppet. Det första gav vaga träffar medan att söka på det andra gav direkta träffar som ofta pekade rakt mot Tom och Mary Poppendieck.

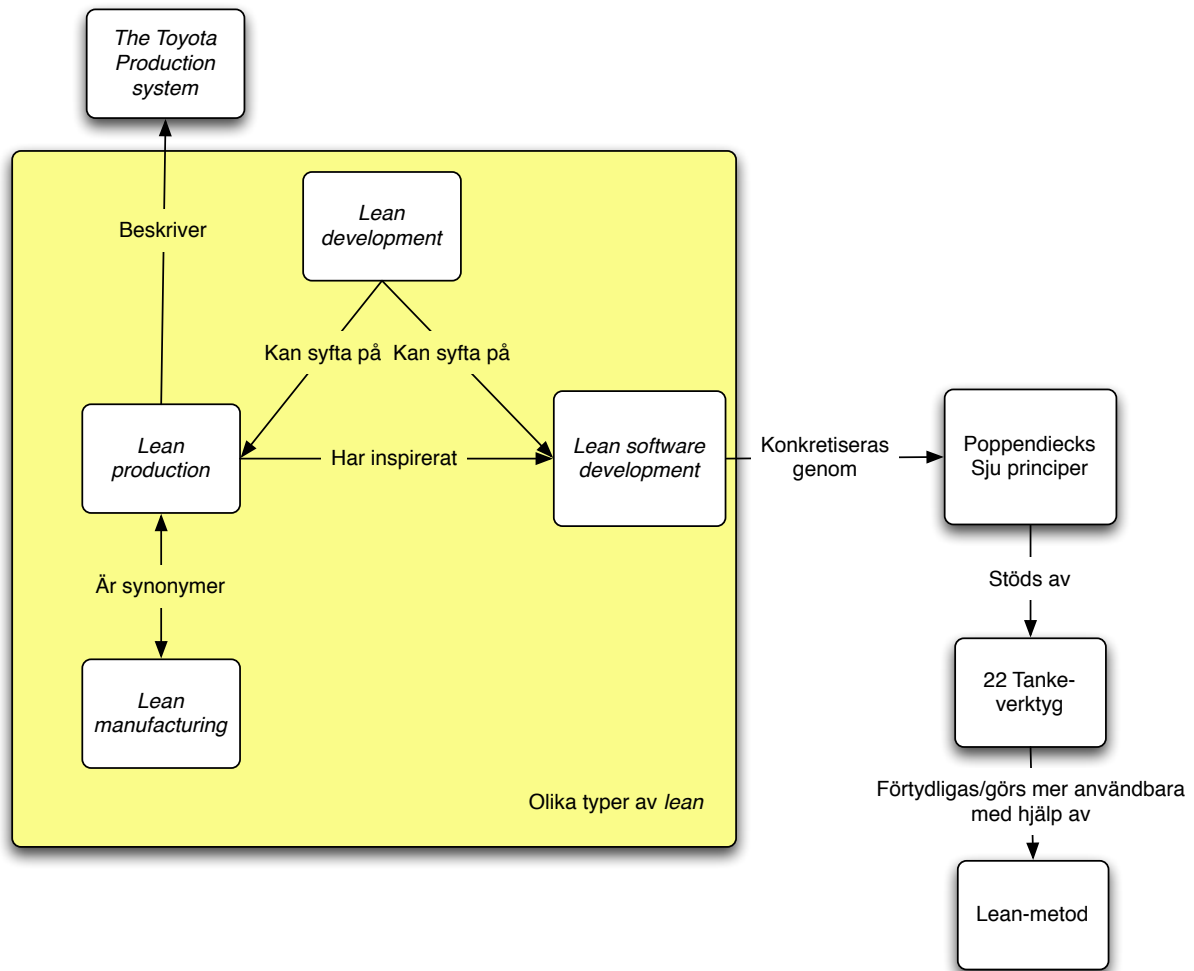
Poppendieck & Poppendieck (2003) använder i sin bok begreppet *lean development* som synonym till *lean software development* men de använder det även för att beskriva hur produktutveckling skedde på Honda och Toyota på 1980-talet. Följande två citat beskriver begreppsförvirringen.

Lean Development further expands the theoretical foundations of agile software development. But it goes further by providing thinking tools to help translate lean principles into agile practices that are appropriate for individual domains (Poppendieck & Poppendieck, 2003, s.xxiii)

Här tolkar jag det som att Poppendieck & Poppendieck’s (2003) talar om *lean development* som en beskrivning av sin egen bok ”*Lean software development – an agile toolkit*”. Boken utgörs av 22 tankeverktyg som tillåter en organisation att anlägga principerna inom *lean production* i systemutvecklingsdomänen.

The approach to product development exemplified by Honda and Toyota in the 1980s, typically called *lean development*, was adopted by many automobile companies in the 1990s (Poppendieck & Poppendieck, 2003, s.xxv)

Här används *lean development* för att beskriva det Womack et al. (1990) kallar *lean production*. Jag har hädanefter valt att betrakta begreppen på det sätt som Figur 1 2 beskriver dem.



Figur 3 - Klargörande modell över begrepp

Lean production/lean manufacturing beskriver båda de principer som ”The Toyota Production system” utgörs av. *Lean development* kan syfta på både *lean production* och *lean software development*.

Lean software development är därmed en beskrivning av hur *lean production* kan användas inom systemutvecklingsdomänen. *Lean software development* konkretiseras av Poppendieck & Poppendieck’s (2003) sju principer vilka stöds av 22 tankeverktyg. Studiens resultat är en metod som förtydligar Poppendieck & Poppendieck’s (2003) principer och tankeverktyg samt gör dem mer användbara genom en arbetsprocess och de mallar som behövs för utförande.

3.3.2 Lean production

För att göra det begripligt vad *lean production* innebär kommer jag i avsnitten 3.1.1.1 – 3.1.1.7 att ge en introduktion till området baserat på Womack et al. (1990) bok ”The machine that changed the world”.

3.3.2.1 The Toyota Production System

Fordonsindustrin i västvärlden hade under 80talet i stort inte förändrat sina arbetsprocesser sen Henry Ford’s massproduktionssystem uppfanns. Efter andra världskriget fick de västerländska biltillverkarna större och större problem att konkurrera med de japanska tillverkarna sätt att producera bilar. På Toyota i Japan utvecklades från 50-talet och framåt ett annat sätt att producera bilar, the Toyota Production System (TPS). Det är från TPS som begreppet *lean production* härstammar eftersom *lean production* är en beskrivning av TPS.

3.3.2.2 Ständig förbättring

Begreppet *lean production* beskrivs av Womack et al. som att använda mindre av allt jämfört med den traditionella massproduktionen med målet satt mot perfektion. De går så långt som att säga att allt skall halveras. Arbete, utrymme, investeringar i verktyg, lagerhållning m.m.

Detta skall uppnås genom kontinuerligt minskande kostnader, noll defekter, noll lager och en oändlig variation av produkter. Denna strävan mot kontinuerlig förbättring kallas *kaizen* och är vad som driver utvecklingen framåt enligt *lean production*.

3.3.2.3 Slöseri

Allt som inte tillför ett värde till slutprodukten ska betraktas som ett slöseri. Det kan gälla slöseri med tid, arbete eller material. Detta slöserikoncept är centralt och kallas *muda*. För att minimera *muda* är det exempelvis viktigt att arbetare inte endast kan utföra en uppgift då de krävs att de kan hjälpa andra när de inte har några arbetsuppgifter att ta itu med.

3.3.2.4 Ledarskapets roll

Istället för förmän som har en rent övervakande roll finns det inom *lean production* istället teamledare. Alla i teamet är ansvariga för flera av produktionsstegen och måste då jobba tillsammans för att så bra som möjligt utföra uppgifterna. Till skillnad från den övervakande förmannen så utför team-ledaren själv arbetsuppgifterna. Saker som städning, kvalitetskontroll och reparationer är en del av teamens uppgifter.

Ett viktigt inslag är att tid avsätts för att låta alla medlemmar i ett team periodiskt tillsammans komma med förslag för hur arbetsprocessen kan förbättras. Beslut genom konsensus är en central del inom *lean production*.

3.3.2.5 Defekter

Att låta defekter först upptäckas när de har gått igenom hela produktionskedjan betraktas som ett slöseri. Det ska undvikas genom att direkt när en defekt upptäcks stanna produktionen till dess att problemet är löst. Womack kallar detta för att ”dra i snöret” eftersom det på Toyota faktiskt installerades ett snöre ovanför det löpande bandet som varje medarbetare kunde dra i för att stanna produktionen. När detta hände så samlades alla för att lösa problemet genom att ställa sig frågan ”varför” fem gånger.

Följande rent hypotetiska exempel kan illustrera en tänkbar kedja av frågor: ”Varför är har dörren defekter? Pressen har en hög felmarginal. Varför har den en hög felmarginal? Pressen

är gammal. Varför är pressen gammal? Avdelningschefen prioriterade att köpa nya verktyg till defekthanteringen. Varför behövdes nya verktyg till defekthantering? Fabriken producerar för många defekter. Varför producerar fabriken för många defekter? Investeringar görs endast för att reducera defekter efter att de har uppkommit”

Lean vänder sig även mot det traditionella supply-chain tänkandet där ingenjörer levererar ritningar på individuella delar till flera tillverkare för budgivning där den med lägst kostnad vinner. Istället levererar man krav på produkter som tillverkare måste möta. Exempelvis kan en tillverkare få till uppgift att producera bromsar med diametern 5x5 som kan stanna en 20 tons bil i 100km/h tio gånger utan minskad bromskapacitet för 40\$. Fungerar prototypen får leverantören jobbet.

3.3.2.6 Pull system

Lean production använder sig av ett just-in-time system kallat *kanban* för att hantera fördelningen av arbetsuppgifter i syfte att minimera lagerhållningen. Inom produktionen innebär det att delar endast ska produceras på föregående steg i flödet för att direkt möta behovet i nästa steg. För att göra detta använde man traditionellt de containrar som transporterade delar. Allteftersom en container blev använd skickades den tillbaka för att indikera att det var dags att göra fler delar. Det system medförde dock att om en enda del fattades så stannade hela produktionen. Men detta var precis poängen, det fokuserade varje deltagare i produktionsprocessen att förutsäga problem innan de blev seriösa nog för att stanna tillverkningen. Detta sätt att fördela arbetsuppgifter kallas för ett pull-system.

3.3.2.7 Kunden

Toyotas *kanban* system började alltid med kunden. Rent praktiskt innebär det att Toyotas återförsäljare inte sålde bilar som ännu inte har producerats eftersom signalen att en kund har beställt en bil blir vad som startar hela tillverkningsprocessen. Detta arbetssätt innebar att återförsäljare jobbade nära fabriken för att se till att de endast sålde bilar som faktiskt kunde produceras. Tanken att matcha försäljning mot produktion visade sig tydligare genom att bilar såldes genom att gå dörr till dörr och anpassa försäljningstaktikerna efter det aktuella behovet hos kunderna.

Toyota fokuserade hårt på att få lojala kunder. Genom att analysera data om kunderna kan de snabbt anpassa produktutbudet för att matcha vad kunderna vill ha när deras olika mönster i livet förändrades.

3.3.3 14 principer för lean production

Jeffrey K Liker har under mer än 20 års tid studerat Toyota och företag som lär av Toyota. Utifrån detta arbete har han sammanfattat 14 principer för *lean production* vilka delas in i 4 kategorier (Liker, 2004). Dessa principer är inte att förväxla med TPS. TPS är det bäst fungerande och mest välutvecklade exemplet på vad Liker's (2004) principer kan åstadkomma.

Att i detta avsnitt redogöra för Liker's (2004) principer är användbart för att längre fram (se avsnitt 6.1) se hur Poppendieck & Poppendieck's (2003) principer förhåller sig till Liker's (2004) principer för *lean production*.

Tabell 1 - Liker's 4P och 14 principer

Kategori	Princip
Philosophy	1. Basera beslut på långsiktig filosofi, inte kortsiktiga finansiella mål
Process	2. Skapa ett kontinuerligt processflöde som lyfter problem till ytan 3. Använd pull-system för att undvika överproduktion 4. Jämna ut arbetsbelastningen 5. Skapa en kultur som stannar upp för att laga problem, gör det rätt från första början 6. Standardiserade uppgifter är grunden för ständigt lärande och bemyndigande av medarbetaren 7. Använd visuell kontroll för att inte låta problem vara dolda 8. Använd endast pålitlig, väl beprövad teknologi som stödjer medarbetare och processer
People/Partners	9. Skapa ledare som förstår arbetet, lever filosofin och lär ut den till andra 10. Skapa exceptionella människor och team som lever efter filosofin 11. Respektera nätverket av partners och leverantörer genom att utmana dem och hjälpa dem växa
Problem solving	12. Gå och se själv för att verkligen förstå situationen (Genchi genbutso) 13. Ta beslut långsamt och med konsensus, betänk noga alla alternativ och implementera beslut snabbt 14. Bli en lärande organisation genom obeveklig reflektion (Hansei) och ständig förbättring (Kaizen)

Tabell 1 visar Liker's (2004) kategorier, enligt honom kallade 4P, under vilka principerna har grupperats kring.

För att bättre kunna förstå vad Liker (2004) anser vara de grundläggande principerna inom *lean production* ges här sammanfattande övergripande beskrivning av hur han beskriver varje princip.

1. Basera beslut på långsiktig filosofi, inte kortsiktiga finansiella mål. Det filosofiska ändamålet måste ligga som grund för beslut istället för kortsiktiga finansiella mål. Exempelvis är det bättre att göra investeringar som lönar sig över flera år istället för att endast tänka på nästa kvartalsrapport. Organisationen måste kämpa mot att bli bättre genom att leverera högre värde för kunder, samhället och samhällets ekonomi.
2. Skapa ett kontinuerligt flöde som lyfter problemen till ytan. Designa arbetsprocesser så att material och information flödar snabbt och gör problem synliga direkt
3. Använd pull-system för att undvika överproduktion. Kunder ska få det de vill ha, när de vill ha det och i den kvantitet de vill ha det. Möt variationer i kunders behov istället för att schemalägga produktionen.
4. Jämna ut arbetsbelastningen (heijunka). Ojämn arbetsbelastning måste undvikas. Arbete ska ske kontinuerligt, inte i stora klumpar.
5. Skapa en kultur som stannar upp för att laga problem, gör det rätt från första början. Arbetsredskap ska stanna sig själva vid problem. Problem ska belysas för team och ledare med hjälp av visuella verktyg. Bygg in supportsystem som gör att det snabbt går att lösa problem och förhindra att de uppstår igen. Kulturen ska sikta mot att avstanna eller sakta ner arbetet för att säkra kvalitén från första början.
6. Standardiserade uppgifter är grunden för ständigt lärande och bemyndigande av medarbetaren. Använd stabila, reproducerbara metoder där det är möjligt för att upprätthålla en hög grad av förutsägbarhet, timing och output från processen. Tillåt medarbetare att vara med och utveckla dessa standarder. Standarder ska vara gjorda på ett sådant sätt att de går att lämna över till en ny medarbetare om en slutar.
7. Använd visuell kontroll för att inte låta problem vara dolda. Använd enkla visuella indikatorer som hjälper människor att se om de befinner sig inom vad som är normalfallet eller om de avviker. Undvik datorskärmar när det inte behövs för att låta medarbetare fokusera. Använd enkla system vid arbetsplatserna för att möjliggöra flöde och "pull"-systemen. Håll längden på rapporter som skrivs begränsade till max ett papper.
8. Använd endast pålitlig, väl beprövad teknologi som stödjer medarbetare och processer. Använd teknologi för att stödja medarbetare, inte ersätta dem. Genomför tester av ny teknik innan den används för att säkra hur väl den fungerar i flödet. Byt ut teknik som inte passar in i organisationen eller stör flödet. Uppmuntra medarbetare att tänka på ny teknik, implementera den fort när den har testats.
9. Skapa ledare som förstår arbetet, lever filosofin och lär ut den till andra. Skapa ledare inom organisationen istället för att anskaffa dem externt. Det är viktigt att ledaren för organisationen förstår arbetet som utförs i detalj.

10. Skapa exceptionella människor och team som lever efter filosofin. Skapa en stark kultur som baseras på organisationens värderingar och lever dem fullt ut. Använd korsfunktionella team för att höja kvalitet och produktivitet. Sträva efter att lära individer hur man arbetar i team.
11. Respektera nätverket av partners och leverantörer genom att utmana dem och hjälpa dem växa. Partners ska ses som en förlängning av ens egna företag. Genom att pusha och utmana dem hjälper man dem framåt.
12. Gå och se själv för att verkligen förstå situationen (Genchi genbutso). Gå till källan och observera vad som händer eller har hänt för att verkligen förstå ett problem istället för att förlita sig på rapporter eller andrahandsinformation.
13. Ta beslut långsamt och med konsensus, betänk noga alla alternativ och implementera beslut snabbt. Betänk olika alternativ så länge som det går, när ett välgrundat beslut har tagits ska det verkställas fort men varsamt. Nå konsensus vid beslut (Nemawashi) eftersom det möjliggör välgrundade beslut. Att nå konsensus tar tid men lönar sig i längden då problemen verkligen har belysts.
14. Bli en lärande organisation genom obeveklig reflektion (Hansei) och ständig förbättring (Kaizen). Genom att lösa problem vid källan och förhindra att de uppstår igen möjliggörs denna ständiga förbättring. Detta gäller även slöserier (muda). Se till att medarbetare lär sig mer och mer och befodra dem långsamt med noggrannhet. Reflektera vid målstolpar och efter projekt kring problem som uppstått och hur de har lösts. Hitta sätt att se till att de inte kan hända igen.

3.3.4 Lean software development

Det finns enligt Morgan (1998) övertygande bevis för att principerna inom *lean production* är applicerbara på systemutveckling. Detta styrks även av Raman (1998) och Morgan (1943).

Systemutveckling har flera likheter med traditionell tillverkning i exempelvis bilindustrin (Cook & Semouchtchak, 2004). I traditionell systemutveckling betonas att analys och beslut skall tas tidigt i utvecklingsprocessen. Jämför detta med traditionell tillverkning där tillverkare tidigt beslutar exakt om hur en bil ska produceras. Tiden delarna spenderar på bandet syftar endast till att sätta samman delarna enligt denna plan. (Womack et al., 1990)

Mary och Tom Poppendieck (2003) beskriver dock att det finns en inneboende skillnad mellan tillverkning och systemutveckling. Systemutveckling är en tankeprocess som mer kan liknas vid hur en kock utformar ett recept, tillverkning kan anses vara att följa detta recept.

Tabell 2 - Skillnad mellan tillverkning och systemutveckling

Systemutveckling	Tillverkning
Kvalitet är lämplighet för användning	Kvalitet är konformism till krav
Varierande resultat är något bra	Varierande resultat är dåligt
Iterationer genererar värde	Iterationer (efterbehandling) genererar slöseri

Detta är något vi måste ha i åtanke när vi inför *lean* i en ny domän, i detta fall systemutveckling.

Mjukvaruutveckling har enligt Mary och Tom Poppendieck (2003) ofta inte tagit åt sig av den förändring som har skett i tillverkningsindustrin de senaste decennierna utan lever ofta kvar i gamla mönster från massproduktionen. Under tiden för TPS framväxt hade bilföretag i USA anammat tanken om att det bästa sättet att undvika kostnader är att ta rätt beslut så tidigt som möjligt. Toyota tog ett annat perspektiv och hävdade att det inte är besluten som är problemet utan det faktum att de inte går att ändra på beslut som har tagits. Enligt Toyota bör beslut tas så sent som möjligt i processen för att besluten då tas med bästa möjliga informationsunderlaget. (Womack et al., 1990)

För att vara framgångsrik med *lean*, oavsett om det är i produktion eller utveckling, hävdar Tom och Mary Poppendieck (2003) att en kulturförändring måste komma till stånd. Först när organisationen har tagit till sig de grundläggande filosofiska grundpelarna i *lean* kan de framgångsrikt arbeta med det. Detta är något som deras principer och tankeverktyg också behandlar genomgående. Påståendet stärks även av Cook & Semouchtchak (2004) samt Middleton (2001). Middleton (2001) menar att om det finns en vilja att förändra organisation så kan *lean* tekniker guida och accelerera förändringarna som behövs genom att snabbt visa på de barriärer som organisationen har satt upp vilka förändrar detta.

Det finns enligt Poppendieck & Poppendieck's (2003) två krav för att en idé ska få fäste i en organisation vilka de möter genom sina principer och tankeverktyg.

1. Det måste framkomma att idén fungerar operationellt
2. Människorna som vill anamma förändringen måste förstå varför den fungerar

3.4 Metodbegreppet

3.4.1 Allmänt om metoder

Metodbegreppet är centralt i denna studie eftersom resultatet syftar till att producera en metod för *lean software development*. För att förstå vad som menas med metod måste detta därför definieras.

Jayaratna (1999) definierar metod som ett explicit sätt att strukturera tankar och handlingar. Metoder innehåller enligt Jayaratna (1999) modeller och speglar ett visst sätt att se på verkligheten baserat på underliggande filosofiska paradig. En metod visar vad, hur och framförallt varför en användare av en metod måste följa de steg som metoden beskriver. Jayaratna (1999) hävdar att en metod skiljer sig från ett ramverk genom att en metod alltid lägger ut steg som går att härleda till olika punkter i tiden medan ramverk befinner sig på en meta-nivå som kan användas på flera olika sätt.

Brinkkemper (1996) definierar metod som ett tillvägagångssätt för att utföra ett systemutvecklingsprojekt genom ett specifikt sätt att tänka vilket innefattar vägledning och regler för hur aktiviteter inom systemutveckling ska utföras.

Goldkuhl (1991) definierar metod som vägledningar för det praktiska arbetet med systemutveckling vilka innehåller riktlinjer för analys/utformning och beskrivning. En metod är vad som bestämmer systemutvecklingens arbetssätt.

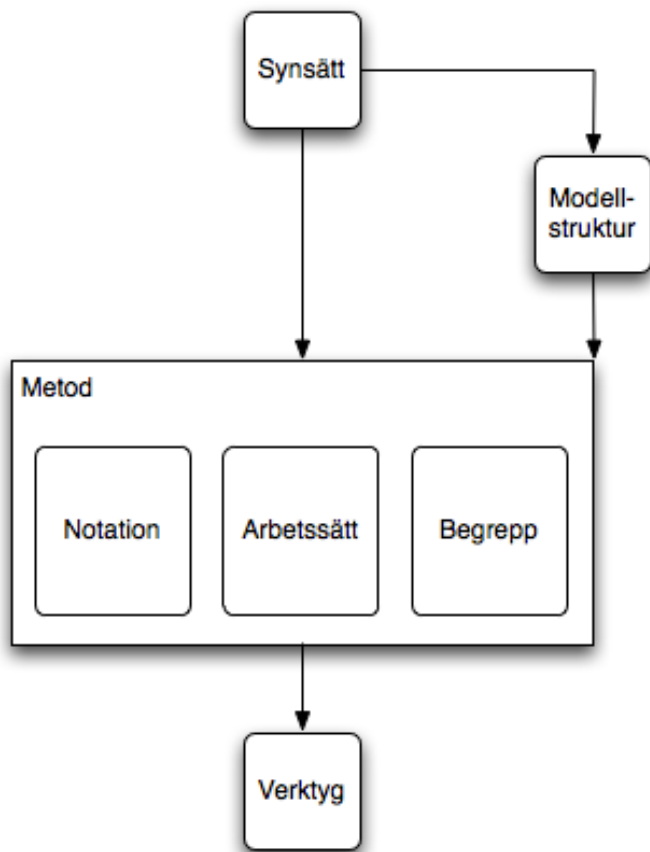
Ofta innehåller metoder regler för olika typer av beskrivningar. En metod påverkar hur man uppfattar verkligheten inom systemutvecklingsprocessen. Genom att använda sig av en metod i processen styr den hur vi ser på verkligheten och med vilka begrepp vi beskriver den. (Goldkuhl, 1991)

För att en metod ska vara användbar behövs det en form av notation som definierar hur något skall beskrivas i en metod och på vilket sätt det ska dokumenteras. Denna notation består enligt Goldkuhl (1991) av:

1. Semantik (Vilka beskrivningselement)
2. Uttrycksform (Hos beskrivningselementen)
3. Syntax (Regler för kombinationer av beskrivningselement)

Goldkuhl (1991) säger därför att en metod består av tre delar: arbetssätt, notation och begrepp.

Metodbegreppet har enligt Goldkuhl (1991) även ett antal närliggande begrepp som är relevanta för att beakta för att förstå begreppet metod som helhet. För att förtydliga vad som menas när metod används i studien är det därför viktigt att förstå även dessa närliggande begrepp. I sin artikel ritar Goldkuhl (1991) upp en metodstruktur som enligt mig väl sammanfattar vad som ingår i begreppet. Nedan görs därför en beskrivning av metodstrukturen och de begrepp som inte redan definierats. Denna metodstruktur kommer att användas för att validera om studiens resultat går att betrakta som en metod.



Figur 4 – Goldkuhl's (1991) metodstruktur

3.4.2 Synsätt

Ett synsätt behöver inte vara särskilt tydligt i samband med modellen eller metoden enligt Goldkuhl (1991).

Enligt Goldkuhl (1991) innehåller ett synsätt principer, värderingar, föreställningar, erfarenheter, kategoriseringar och definitioner som är relevanta för systemutveckling.

3.4.3 Modell och modellstruktur

Goldkuhl (1991) definierar modell som en övergripande struktur för systemutvecklingsprocessen. Modeller definierar *vad* som ska göras i systemutvecklingsprocessen men definierar inte *hur* det ska göras. Modeller är dock ofta till viss grad beroende av metoder som definierar hur något ska göras. Goldkuhl (1991) hävdar därför att modeller inte är strikt oberoende av metoder utan kan ha olika grader av metodberoende. Modeller bestämmer vilka olika faser, dvs arbetsuppgifter eller klasser av arbetsuppgifter, som ingår i systemutvecklingsprocessen.

Faser i systemutvecklingen grupperas tillsammans ihop i en modellstruktur som definierar uppdelningen i områden/faser som hela eller delar av systemutvecklingen består av. (Goldkuhl, 1991)

3.4.4 Verktyg

Enligt Goldkuhl (1991) är verktyg något som stödjer analys- och konstruktionsprocessen under systemutveckling. Exempelvis datorbaserade programvaror.

3.5 Agila metoder

Det är svårt att hitta en enhetlig definition av vad agila metoder är eftersom det mesta som är skrivet inom ämnet är gjort av praktiker eller konsulter istället för akademiker (Abrahamsson et al., 2002). Abrahamsson (2002) sammanfattar begreppet agilt genom att beskriva de värden som utövare av agila metoder bekänner sig till:

- Individer och interaktion över processer och verktyg
- Fungerande program över uttömmande dokumentation
- Samarbete med kunden över kontraktsförhandlingar
- Anpassning till förändring över att följa en plan

3.6 Systemutveckling

Systemutveckling innebär i grunden att använda olika slags beskrivningar för att skapa ett nytt informationssystem. Kravspecifikationer är beskrivningar av funktionalitet som önskas, analyser är beskrivningar av ex konsekvenser av att införa ett nytt informationssystem och systemutvecklingsmetoder är beskrivningar över hur systemutvecklingen ska gå till. För att skapa ett nytt informationssystem använder utvecklare i processen en eller flera olika systemutvecklingsmetoder. (Andersen, 1994a)

3.7 Organisation

Organisationer är instrument skapade för att nå mål (Morgan, 1943). Enligt detta perspektiv av organisationer som verktyg beskriver Røvik (1998) en organisation som ”redskap för att effektivt åstadkomma beslut, varor, åtgärder och tjänster”.

Detta är det organisationsperspektiv som ligger närmast den typ av organisation som diskuteras i befintlig teori (Womack et al., 1990) och därför även det perspektiv som denna studie kommer att anta på begreppet.

När jag talar om systemutvecklingsorganisationer i studien är det ett sätt att beskriva en verksamhet som primärt har som mål att genomföra systemutvecklingsprojekt.

4 Empiri

4.1 Informella konversationer

För att skapa en metod för *lean software development* har åtta informella konversationer genomförts med föreståndaren för systemutvecklingsorganisationen InnovationLab. Kapitlet redogör för dessa möten genom att redovisa minnesanteckningar från möten och hur dessa har fungerat som underlag till skapandet av en metod för systemutveckling med *lean software development*.

2011-03-04 – Konversation 1

Tema på diskussionen: Lean inom systemutveckling

Den första informella konversationen var ett inledande möte där vi diskuterade övergripande kring hur lean kan användas i olika typer av organisationer och då framförallt inom systemutvecklingsorganisationer. Mötet var ett sätt för mig att införskaffa initiala tankar om hur en praktiker ser på lean inom systemutveckling.

2011-03-17 – Konversation 2

Tema på diskussionen: Nuvarande arbetsprocess på InnovationLab, Standardisering av arbete, Författare som skrivit om lean, Lean inom systemutveckling, Traditionell användning av Lean, Agila metoder, Organisationsförbättring, Effektivisering av arbete, Dokumentation, Lean-metod

Vi diskuterade hur arbetssituationen ser ut på InnovationLab idag. Idag arbetar InnovationLab med många olika projekt, som alla sker på olika sätt. Föreståndarens uppfattning är att detta inte känns riktigt rätt och föreståndaren vill därför standardisera arbetet. Uppfattningen som föreståndaren har kring lean är att det är ett sätt att arbeta mot standardisering.

Vi diskuterade olika källor och kom fram till att en bra utgångspunkt är Liker's Toyota Way och Womack's The Machine that Changed The World eftersom de är väletablerade och vi både känner till dessa.

Frågan om varför lean är intressant diskuterades. Nuvarande systemutvecklingsprojekt har försökt att använda olika agila metoder, nu vill föreståndaren förbättra hela organisationen inte endast ett enskilt projekt. Allt från hur organisationen hanterar kundrelationer till hur de effektivt hanterar olika typer av förfrågningar och projekt samt hur specifika projekt hanteras sinsemellan är områden som är intressanta för föreståndaren att behandla inom en ny metod.

Föreståndaren tror att lean skulle kunna passa eftersom det är organisationsövergripande och behandlar frågor såsom standardisering och effektivisering.

Föreståndaren pratar inte så mycket om olika verktyg för lean men pratar om de 14 principerna som Liker tycker Toyotas modell består av. Vi diskuterade om hur det går att göra om dem så att de passar en tjänsteorganisation. Föreståndaren menar att för att göra detta behöver man ha fantasi. Vi diskuterade om att det säkert finns många som har provat det och att jag tar fram underlag över läget.

Grundproblemet som föreståndaren beskriver det är att alla arbetar på olika sätt, vill standardisera. Vi diskuterade frågan hur vi får till något som är ett mellanting mellan att ha allt dokumenterat och inget dokumenterat.

Föreståndaren menar att om man inte kan standardisera saker kommer de att glömma bort sakerna och det hamnar i ett svart hål. Föreståndaren vill behålla kompetensen och förbättra organisationen.

Några interna mål som föreståndaren beskrev var att alla ska vara minst 85 % belagda. Förra året var den genomsnittliga beläggningsgraden runt 98 %.

InnovationLab har inga ekonomiska mål eftersom de inte får gå med vinst, de tillhör nämligen en statlig myndighet. Vi kom fram till att detta kan vara svårt att ta med i lean-metoden utan mer är något att tänka på att lean-metoden ska kunna fungera under dessa situationer.

Vi pratade om hur de 5 stegen som Womack beskriver kunde vara intressanta för InnovationLab.

- Definiera kundvärdet
- Definiera värdeflödet
- Få det att flyta jämt
- Tillverka mot kundorder
- Alltid sträva mot högsta kvalitet

2011-03-18 – Konversation 3 & 2011-03-28 – Konversation 4

Tema på diskussionen: Nuvarande arbetsprocess på InnovationLab

För att bättre förstå hur arbete bedrivs inom en systemutvecklingsorganisation spenderade vi ett flertal timmar till att genomföra en modellering av den nuvarande arbetsprocessen på InnovationLab. Denna har fungerat som ett underlag vid skapandet av metoden på det sätt att det vid senare diskussioner har gått att relatera diskussionsmaterial till arbetsprocessen för att se hur förändringar förhåller sig till den nuvarande arbetsprocessen. Studien resultat är dock inte knutet till denna arbetsprocess men det kan vara intressant för läsaren att veta hur den har fungerat som underlag vid de informella konversationerna.

Arbetsprocessen modellerades genom att vi tillsammans ritade på ett utkast till en modell av verksamhetens arbetsprocess. Detta gjordes genom att vi utgick från projekt med en generell timeline och identifierade varje aktivitet som skedde inom detta.

Exempel på resonemang som diskuterades under modelleringen för att hitta aktiviteter:

Författaren: När ni tar ett möte med tekniker hur vet du vem du ska prata med?

Föreståndaren: Jag vet i huvudet hur folk är belagda, kolla vi ritar in det så här...

Författaren: Hur hämtar ni in kundönskemål?

Föreståndaren: Tar ett möte 1 timme, just ja vi måste rita in avtalsprocessen också i modellen.

Författaren: Ok när gör ni avtal?

Föreståndaren: När det är mindre omfattning än 40 timmar. Fördelningen mellan projekt är 70/30 interna - externa, 2009 50/50, 2010 minskade projekten totalt.

Författaren: Menade du då att dra linjen mellan dessa objekt på detta sätt?

Föreståndaren: Ja precis så menade jag.

2011-04-04 – Konversation 5

Tema på diskussionen: Lean-metod, Nuvarande arbetsprocess på InnovationLab, Poppendieck & Poppendieck's principer och tankeverktyg, lean software development, framtida arbetsprocess

Diskuterade kring verktyg som föreståndaren hade hittat och som han tyckte kunde vara intressanta att ta med i processen. Jag hävdade att dessa saker är något som kommer kunna användas för att utföra det resultat som studien visar och därför inte behöver vara med direkt i studien.

Vi gick därefter igenom arbetsprocessmodellerna ännu en gång för att säkra att vi har uppfattat hur situationen nu ser ut. Detta gav en förståelse för hur verksamheten fungerar och kändes nyttigt att ha genomfört för att bättre förstå hur lean-metoden bör skapas.

Efter detta gick vi igenom Poppendieck & Poppendieck's (2003) principer och verktyg muntligt för att se om dessa kunde passa på InnovationLab, enades om att jag skapar ett underlag som vi kan gå igenom i större detalj på nästa möte.

2011-04-14 – Konversation 6 & 2011-04-21 – Konversation 7

Tema på konversation 6: Styrkor med Poppendieck & Poppendieck's principer och tankeverktyg, problem avseende användning av Poppendieck & Poppendieck's principer och tankeverktyg

Tema på konversation 7: Relevans kring Poppendieck & Poppendieck's principer och tankeverktyg för InnovationLab, Förslag till förändring av Poppendieck & Poppendieck's principer och tankeverktyg, Förslaget till Lean-metod

Poppendieck & Poppendieck's principer och tankeverktyg gick noggrant igenom och diskuterades. Resultatet från denna diskussion finns bifogat som bilaga I. Detta är den aktivitet som var av störst betydelse för att kunna skapa studiens resultat. Genom att ha diskuterat de bakomliggande teorierna och förutsättningarna i detalj kunde dessa två informella konversationer genomföras enkelt då båda förstod vad som menades med varje del. I dessa konversationer validerades det teoretiska materialets lämplighet som underlag för en metod för *lean software development*. Som framgår av bilaga I så var det väldigt få saker i materialet som behövde förändras för att passa föreståndarens organisation.

2011-05-13 – Konversation 8

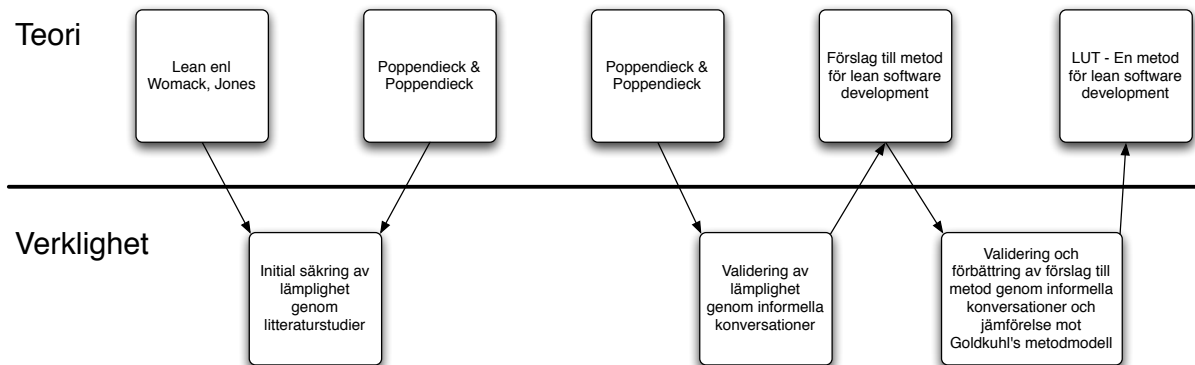
Tema på diskussionen: Förslaget till lean-metod

Under mötet gick vi igenom och gjorde smärre textuella revideringar av mallarna (se bilaga A-F). Strukturen och innehållet förblev i stort sett lika mot det underlag som diskuterades.

Vi kom fram till enkla saker såsom att byta plats på krav och värde i förstudiemallen och att mallarna kan läsas sekventiellt. En uppdatering i färgkoder som skiljer fält åt behövde göras.

4.2 Det abduktiva arbetssättet

Avsnittet presenterar en figur som illustrerar hur abduktion har använts inom studien.



Figur 5 - Studiens abduktiva arbetssätt

Först genomförs en teoretisk studie som säkrar att koncepten lean software development och lean production i grunden använder sig av samma principiella utgångspunkter.

Efter detta valideras Poppendieck & Poppendieck's (2003) principer och tankeverktyg i praktiken genom informella konversationer med föreståndaren. Detta möjliggör skapandet av ett förslag till metod för *lean software development*.

Denna valideras genom informell konversation med föreståndaren för InnovationLab och ger upphov till smärre förändringar vilket möjliggör skapandet av den slutgiltiga metoden LUT – lean systemUTveckling.

5 Poppendieck's principer och tankeverktyg

5.1 Principer

Enligt Poppendieck & Poppendieck (2003) finns det sju principer för *lean software development* och 22 tankeverktyg som kan användas för att skapa en agil metod. Som ett led i arbetet med att besvara studiens forskningsfrågor har en analys av dessa principer och tankeverktyg genomförts. Detta kapitel ger en kort sammanfattning kring principerna och tankeverktygen i syfte att introducera läsaren till Poppendieck & Poppendieck's (2003) tankar om vad som bör ligga till underlag för skapandet av en metod för *lean software development*.

Tabell 3 - Poppendieck's 7 principer för *lean software development*

Princip	Beskrivning
Eliminera slöseri (muda)	Allt som inte tillför värde till produkten/kunden just nu är slöseri
Förstärk lärandet	Behåll kunskap för att möjliggöra välgrundade beslut
Ta beslut så sent som möjligt	Vänta så länge som möjligt för att ta så väl underbyggda beslut som möjligt
Leverera så snabbt som möjligt	Produkter ska levereras så snart det är möjligt
Bemyndiga teamet	Beslut tas av den som har kunskapen oberoende av vilken plats i organisationen den befinner sig på
Bygg in integritet	Beakta alltid integritetsaspekter
Se helheten	Beslut ska tas utifrån en helhetssyn

5.2 Tankeverktyg

Poppendieck & Poppendieck (2003) säger att de inte har skapat en kokbok utan gett människor verktygen för att utveckla sina egna recept för hur man kan arbeta med *lean software development* inom sin egen organisation. För att göra detta beskriver de 22 tankeverktyg som praktiker kan använda sig av för att skapa en agil metod som tillåter detta. Tankeverktygen är grupperade efter de sju principer som enligt Poppendieck & Poppendieck (2003) finns kring *lean software development* och syftar alla till att visa hur principerna som återfinns ovan kan tänkas användas. Varje tankeverktyg innehåller en beskrivning kring syftet med tankeverktyget och ett antal förslag till handlingar som en systemutvecklingsorganisation kan genomföra för att uppfylla tankarna bakom dessa principer.

Jag beskriver nedan tankeverktygen kortfattat för att det ska gå att förstå på vad studiens resultat är grundat och hur principerna och tankeverktygen återspeglas i detta. För en utförlig sammanfattning av varje princip och tankeverktyg se bilaga I.

Tabell 4 - Poppendieck's tankeverktyg

Verktyg	Syfte
Slöserier	Eliminera slöseri
Mappnings av värdekedjan	Ge organisationen mätinstrument för att effektivisera
Återkoppling	Öka återkopplingstillfällen i systemutvecklingsprocessen
Iterationer	Genomför systemutveckling inkrementellt
Synkronisering	Se till att nyutveckling passar in i det som andra utvecklat
Set-baserad utveckling	Definiera begränsningar
Optioner	Håll flera tänkbara alternativ öppna länge
"The last responsible moment"	Ta beslut så sent som bara möjligt för att säkra att de tas med så gott underlag som möjligt
Djup först vs vidd först	Se flera olika alternativ till problem och gå inte på djupet förens vid rätt tillfälle
Pull system	Använd ett dragbaserat arbete
Köteori	Minimera slöserier som uppkommer av väntan på köer
Förseningens kostnader	Räkna på vad en försening kostar när fler variabler tas i beaktande
Självbestämmande	Sätt utvecklare i fokus och ge dem möjligheten att göra ett bra arbete
Motivation	Motivera utvecklare för att nå bättre resultat
Ledarskap	Understöd utveckling genom passionerat ledarskap och utveckling av medarbetare
Expertis	Kartlägg, införskaffa och underhåll expertis
Upplevd integritet	Förstå att det är kunden som avgör vad upplevd integritet är
Konceptuell integritet	Se till att systemutveckling har helheten som mål
Refactoring	Lyft problem till ytan och lös innan de blir riktiga problem
Testning	Säkerställ att den individuella delen fungerar som den ska
Mätningar	Mät rätt saker och lös problem vid källan
Kontrakt	Se till att inte lägga risken enbart hos en enskild part

6 Resultat

Kapitlet syftar till att besvara studiens huvudsakliga forskningsfråga samt dess delfrågor ifrån avsnitt 1.3. I avsnitt 5.1 besvaras delfråga 1.1 och i avsnitt 5.2 besvaras delfråga 1.2.

För att säkra huruvida metoden går att betrakta som en metod utifrån ett akademiskt perspektiv beskriver jag i avsnitt 5.3 hur metoden förhåller sig till Goldkuhl's (1991) metodstruktur från avsnitt 3.4. Därefter besvaras den huvudsakliga forskningsfrågan i avsnitt 5.4 - 5.5.

6.1 Likheter/skillnader mellan Poppendieck's och Liker's principer

Jag har i detta avsnitt gjort en undersökning av likheter och skillnader mellan Poppendieck & Poppendieck's (2003) principer för *lean software development* och Liker's (2004) principer för *lean production* (se avsnitt 3.3.3).

Detta görs för att om likheterna mellan dem är väldigt stor möjliggör det för mig att dra slutsatsen att de båda behandlar samma problem och därför även har samma styrkor. Slutsatsen är möjlig eftersom jag i avsnitt 3.3.1 visar att *lean software development* baseras på samma principer som *lean production* och att *lean software development* konkretiseras genom Poppendieck & Poppendieck's (2003) principer.

Genom att identifiera problem och styrkor med Poppendieck & Poppendieck's (2003) principer och tankeverktyg identifierar jag därmed samtidigt problem och styrkor med *lean production* och *lean software development*.

För att se vilka likheter och skillnader som finns så görs i avsnitt 5.1.1 – 5.1.14 en genomgång av Liker's (2004) principer där jag för varje princip studerar om någon av Poppendieck & Poppendieck's (2003) principer behandlar samma sak som den aktuella principen.

6.1.1 Basera beslut på långsiktig filosofi, inte kortsiktiga finansiella mål

Denna princip går inte att direkt matcha mot någon av Poppendieck & Poppendieck's (2003) principer men är något som återfinns utspritt över flera av deras principer.

T.ex. talar Poppendieck & Poppendieck's om vikten av att förstå vad det man utvecklar har för värde för kunden och att alltid ha kundens uppfattning om upplevd integritet med sig vid utveckling. Detta är starkt filosofiska ställningstaganden som tillsammans med Poppendieck & Poppendieck's (2003) andra principer indikerar att skapa värde för kunden i slutändan kommer att generera finansiella resultat. Jag uppfattar därför inte den direkta matchningen som något problem utan anser att genom att se kunden som målet med organisationens alla processer så skapas en koppling mellan principerna.

6.1.2 Skapa ett kontinuerligt processflöde som lyfter problem till ytan

Vikten av att eliminera slöseri är central både hos Liker (2004) och Poppendieck & Poppendieck (2003). Vad Poppendieck & Poppendieck (2003) har gjort i sin princip "Eliminera slöseri" är att se till de slöserier som finns inom traditionell tillverkning och matchat dem mot lämpliga slöserier inom systemutveckling. Båda beskriver hur icke värdeskapande aktiviteter i en organisation ska minimeras.

6.1.3 Använd pull-system för att undvika överproduktion

Att använda ett pull-baserat system är ett sätt att undvika köbildning. Denna princip matchar mot Poppendieck & Poppendieck's (2003) princip "Leverera så snabbt som möjligt". Poppendieck & Poppendieck (2003) går inte i detalj in på hur pull-system ska fungera rent praktiskt vilket både kan uppfattas som ett problem men även en möjlighet för organisationen att själv utforma ett arbetssätt som fungerar för dem baserat på de grundläggande tankarna inom denna princip. Båda belyser dock vikten av att använda ett pull-baserat arbetssätt.

6.1.4 Jämna ut arbetsbelastningen

Matchar mot Poppendieck & Poppendieck's (2003) princip "Leverera så snabbt som möjligt" och "Eliminera slöseri". Poppendieck & Poppendieck's (2003) påpekar att det ger större resultat att ha ett jämt flöde istället för att sträva mot full beläggning.

6.1.5 Skapa en kultur som stannar upp för att laga problem, gör det rätt från första början

Genom att kontinuerligt före, i och efter iterationer återkoppla till kund samt genom att göra relevant information tillgänglig genom hela processen matchar Poppendieck & Poppendieck's (2003) denna princip genom sina principer kring "Förstärk lärandet", "Ta beslut så sent som möjligt" och "Bygg in integritet".

Dessa syftar till att låta beslut tas när det finns så gott underlag som möjligt samt att identifiera problem så snart som de uppstår för att direkt åtgärda dem.

6.1.6 Standardiserade uppgifter är grunden för ständigt lärande och bemyndigande av medarbetaren

I principen "Bemyndiga teamet" beskriver Poppendieck & Poppendieck (2003) vikten av att ha uppdaterade kodstandarder vilka skapas av utvecklare själva. I traditionell tillverkning skapas arbetsrutiner av en chef eller förman och medarbetares roll är endast att utföra dessa uppgifter. Liker (2004) och Poppendieck & Poppendieck (2003) beskriver hur man genom att ge människor kontroll över sitt eget arbete kan skapa ett kvalitetstänkande hos varje enskild medarbetare och en glädje i arbetet som medför högre effektivitet och kvalitet samtidigt som kontinuerlig förbättring av processerna skapas.

6.1.7 Använd visuell kontroll för att inte låta problem vara dolda

Inom den traditionella tillverkningen brukar denna princip genomföras i praktiken med hjälp av maskiner som signalerar när problem uppstår. Poppendieck & Poppendieck (2003) beskriver i sin princip "Förstärk lärandet" ingående hur problem skall föras upp till ytan genom kontinuerlig återkoppling till kunden. Genom att göra delvis färdig designinformation tillgänglig för alla relevanta intressenter kan det snabbt visa sig när utvecklingen är på väg åt fel håll och kunden har då möjlighet att dra i bromsen för att styra in utvecklingen på rätt väg igen.

6.1.8 Använd endast pålitlig, väl beprövad teknologi som stödjer medarbetare och processer

Denna princip är svårare att direkt omvandla till systemutvecklingsdomänen då mycket arbete ofta utförs med ny teknik som inte alltid är den mest beprövade. Poppendieck & Poppendieck (2003) beskriver dock denna problematik inom principen "Leverera så snabbt som möjligt" där uppmuntran exempelvis ges till att se hur valet av ett verktyg kommer att påverka tiden fram till leverans, inte enbart verktygets kostnad. Ytterligare exempel på detta

inom samma princip är att inte fokusera på valet av teknologi i första hand utan se till vad det är som ska utvecklas och fundera över vilka krav detta ställer på valet av teknik. När bredd beaktas på detta sätt blir det enklare att längre fram göra det korrekta valet.

6.1.9 Skapa ledare som förstår arbetet, lever filosofin och lär ut den till andra

Återfinnes väldigt tydligt i Poppendieck & Poppendieck (2003) princip kring ”Bemyndiga teamet” där ledarskapets främsta uppgift är att stödja utvecklingen. Genom att låta team själva fatta beslut över vad som behöver göras i stor grad ställer det krav på att de får så goda förutsättningar för att göra detta som möjligt. Som tidigare beskrivet kommer inte *lean software development* inte att fungera optimalt i en organisation utan att den har tagit till sig de grundläggande värderingarna. Det är ledarnas uppgift att alltid förespråka och visa på värdet i den bakomliggande filosofin och arbetssättet.

6.1.10 Skapa exceptionella människor och team som lever efter filosofin

Vikten av ett ledarskap som kontinuerligt förespråkar filosofin och värdesätter expertis, motivation och självbestämmande beskrivs av Poppendieck & Poppendieck (2003) med principen ”Bemyndiga teamet”. De exceptionella människorna betonas speciellt genom att beskriva hur viktigt det är att organisationen låter *master developers* uppstå.

6.1.11 Respektera nätverket av partners och leverantörer genom att utmana dem och hjälpa dem växa

Poppendieck & Poppendieck (2003) beskriver relationen till partners och leverantörer utifrån frågeställningen hur tillit skall skapas mellan olika aktörer som är involverade i systemutvecklingsprocessen. Där Liker (2004) förespråkar ett aktivt arbete med att involvera sig i och förbättra dessa beskriver Poppendieck & Poppendieck (2003) vikten av kontrakt som inte lägger hela risken hos ena parten i principen ”Se helheten”.

Genom att dela på risken skapas ett intresse hos båda parter att utföra arbetet på bästa sätt medan kontrakt skrivna som skyddsmekanismer för kund eller utförare fokuserar på att beskydda det arbete som de själva utför.

Både Liker (2004) och Poppendieck & Poppendieck (2003) beskriver ett problem med att producenten i första hand är intresserad av att få ett uppdrag vilket har en minsta tillåten nivå som skall uppfyllas.

Genom att använda principerna får vi en styrka som möjliggör för producenten gå ifrån att endast tillverka det som har beställts till att även se till vad det är beställaren behöver.

6.1.12 Gå och se själv för att verkligen förstå situationen (Genchi genbutso)

Poppendieck & Poppendieck (2003) beskriver inte detta ingående och låter det inte få en framträdande roll i sina principer vilket jag uppfattar som ett problem. Att skapa sig en förståelse för problemets egentliga källa genom att ställa frågan varför fem gånger behandlas i Poppendieck & Poppendieck (2003) princip ”Se helheten” men där beskriver de endast hur ledare kan genomföra mätningar för att ta beslut utifrån. Problemet med detta är således att det inte räcker att ge ledare bättre verktyg för att mäta, de måste även förstå det bakomliggande problemet.

6.1.13 Ta beslut långsamt och med konsensus, betänk noga alla alternativ och implementera beslut snabbt

Behandlas väldigt tydligt av Poppendieck & Poppendieck's (2003) principer ”Ta beslut så sent som möjligt”, ”Leverera så snabbt som möjligt”.

6.1.14 Bli en lärande organisation genom obeveklig reflektion (Hansei) och ständig förbättring (Kaizen)

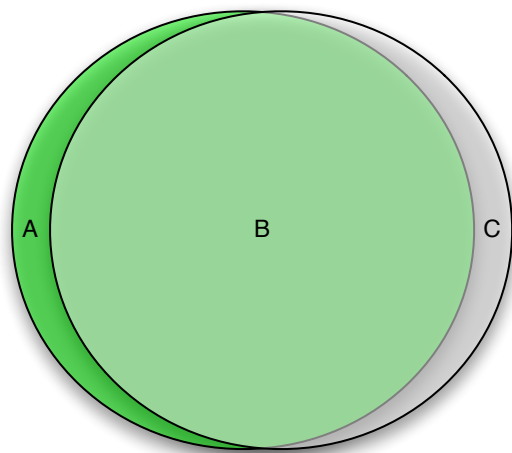
Poppendieck & Poppendieck (2003) beskriver genomgående i alla sina principer vikten av reflektion och ständig förbättring.

6.1.15 Överensstämmelse mellan principerna

Överensstämmelsen mellan Poppendieck & Poppendieck (2003) och Liker (2004) är väldigt stor vilket visar den närhet som *lean software development* har till *lean production*. I de fall en direkt koppling inte kan göras mellan två principer går det ändå att härleda hur Poppendieck & Poppendieck (2003) behandlar det som Liker (2004) beskriver inom ramen för deras andra principer. Det är möjligt att det finns vissa undantagsfall som skiljer de olika principerna åt men det är iså fall en försumbar del vilket illustreras med hjälp av figur 4 där skillnader representeras av A/C och likheter av B.

Exempelvis innehåller A ett problem med att Poppendieck & Poppendieck (2003) inte beskriver vikten av att ledare ska gå ut i organisationen och se problem där de uppstår.

Liker (2004) beskriver i sin tur standardisering av arbetsuppgifter som det starkaste sättet att bemyndiga teamet medan Poppendieck & Poppendieck (2003) betonar självbestämmande som den avgörande faktorn vilket är exempel på en skillnad som illustreras i C.



Figur 6 - Likheter och skillnader mellan Poppendieck's och Liker's principer

Eftersom likheten är så pass stor så är det rimligt att anta att de problem som principerna försöker möta är i stort sett samma i både tillverkningsdomänen och systemutvecklingsdomänen. Det går med bakgrund av detta att visa styrkor och problem med *lean software development* och *lean production* genom en analys av Poppendieck & Poppendieck's (2003) principer och tankeverktyg. Denna analys redovisas i närmare detalj i avsnitt 6.2.

6.2 Problem och styrkor med Poppendieck's principer och tankeverktyg

Poppendieck & Poppendieck's (2003) skriver att deras principer och tankeverktyg ska kunna användas för att skapa en agil metod för *lean software development*.. För att se om detta stämmer behöver en analys göras kring vilka problem som finns avseende beskrivning av principerna och tankeverktygen. Detta genomförs med hjälp av en problemanalys.

För att säkra kvaliteten på metoden behövde jag även förstå på vilket sätt ett användande av Poppendieck & Poppendieck's (2003) principer och tankeverktyg kan vara till nytta för en systemutvecklingsorganisation. Detta gjordes genom en styrkeanalys. När denna analys genomfördes fick jag även en möjlighet att se om några ytterligare problem uppenbarade sig vilka inte möts av några styrkor.

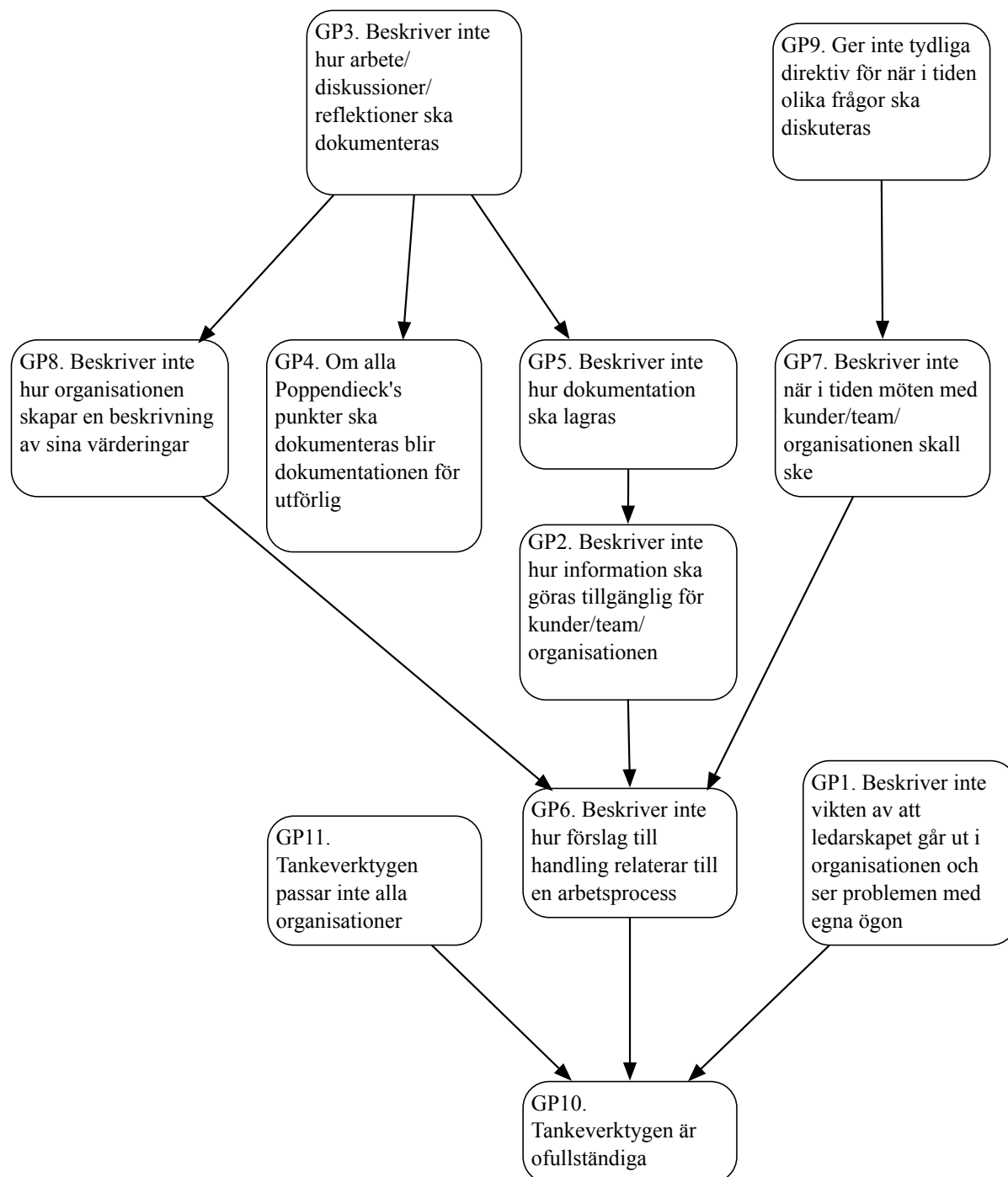
För att genomföra problem och styrkeanalysen har jag analyserat varje princip och dess tillhörande tankeverktyg var för sig för att efter detta diskutera resultatet med föreståndaren för den studerade organisationen. Vid detta arbete gavs en möjlighet att validera att de problem och styrkor som jag har uppfattat vid analysen också uppfattas som problem och styrkor av den studerade organisationen.

Genom att med hjälp av en problemgraf beskriva de identifierade problemen i nästkommande avsnitt ges en tydlig bild över de orsak/verkan-samband som problemen innehar. Varje problem kan ha en eller flera orsaker och verkan vilka alla samlas i ett gemensamt övergripande problem.

Ett av de grundläggande problemen med beskrivningen av Poppendieck & Poppendieck's (2003) verktyg/principer är att de säger vad som ska göras. Det är dock ibland svårt att förstå hur det ska skapa ett värde för organisationen utan att även beskriva hur och när de bör göras i tiden.

De generella problem som beskrivs i kapitel 6.1 har använts som komplement till underlag vid problem och styrkeanalysen.

6.2.1 Problem med beskrivningen av principer och tankeverktyg



Figur 7 - Problemgraf kring beskrivningen av Poppendieck's principer och tankeverktyg

- GP1. Beskriver inte vikten av att ledarskapet går ut i organisationen och ser problemen med egna ögon
- GP2. Beskriver inte hur information ska göras tillgänglig för kunder/team/organisationen
- GP3. Beskriver inte hur arbete/diskussioner/reflektioner ska dokumenteras
- GP4. Om alla Poppendieck & Poppendieck's punkter ska dokumenteras blir dokumentationen för utförlig
- GP5. Beskriver inte hur dokumentation ska lagras
- GP6. Beskriver inte hur förslag till handling relaterar till en arbetsprocess
- GP7. Beskriver inte när i tiden möten med kunder/team/organisationen skall ske
- GP8. Beskriver inte hur organisationen skapar en beskrivning av sina värderingar
- GP9. Ger inte tydliga direktiv för när i tiden olika frågor ska diskuteras
- GP10. Tankeverktygen är ofullständiga
- GP11. Tankeverktygen passar inte alla organisationer

Utgångspunkten tas i problemet med att Poppendieck & Poppendieck's (2003) saknar riktlinjer för hur arbete/diskussioner/reflektioner ska dokumenteras. Eftersom det saknas direktiv så finns det exempelvis i principerna och tankeverktygen endast direktiv för vad som ska dokumenteras. Det som saknas är en beskrivning kring hur resultatet ska dokumenteras och direktiv för hur detta ska synliggöras för relevanta intressenter. Även frågan kring hur det dokumenterade skall hanteras i en arbetsprocess är obesvarad.

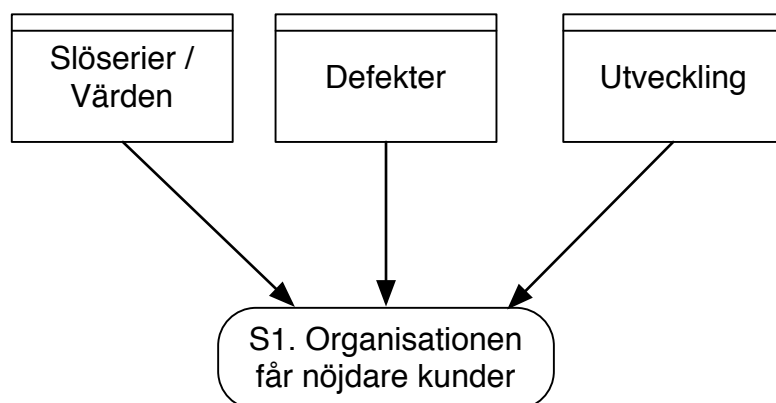
Vid problemanalysen framkom det även ett antal problem som avser beskrivningen och utformningen av Poppendieck & Poppendieck's (2003) principer och verktyg. Dessa gäller framförallt frågor kring dokumentation, relationen till en arbetsprocess, synliggörande av information samt ledarskap.

6.2.2 Styrkor med Poppendieck's principer

Styrkorna som beskrivs nedan illustrerar på vilket sätt en systemutvecklingsorganisation kan förbättras genom att använda Poppendieck & Poppendieck's (2003) principer och tankeverktyg. Beskrivningen av principerna och tankeverktygen grundar sig i att det finns ett antal problem som bör lösas i en systemutvecklingsorganisation. De problem som principerna och tankeverktygen syftar till att lösa är vad som är deras styrkor.

6.2.2.1 Nöjdare kunder

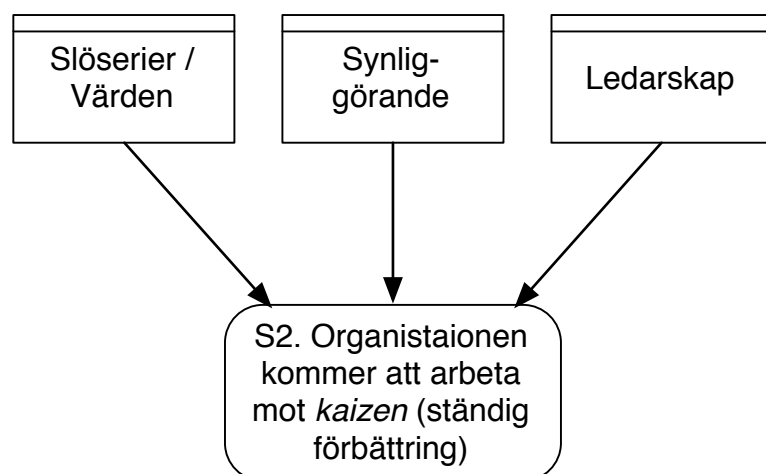
Genomgående inom *lean software development* betonas vikten av att sätta kunden i centrum. Det är därför inte helt orimligt att se nöjdare kunder som en grundläggande styrka med Poppendieck & Poppendieck's (2003) principer och tankeverktyg.



Figur 8 - Styrkegraf avseende nöjdare kunder

Den andra centrala styrkan som finns inom Poppendieck & Poppendieck's principer och tankeverktyg är att ett användande möjliggör för organisationen att arbeta mot ständig förbättring (se kapitel 4 för en djupare genomgång av vad ständig förbättring innebär).

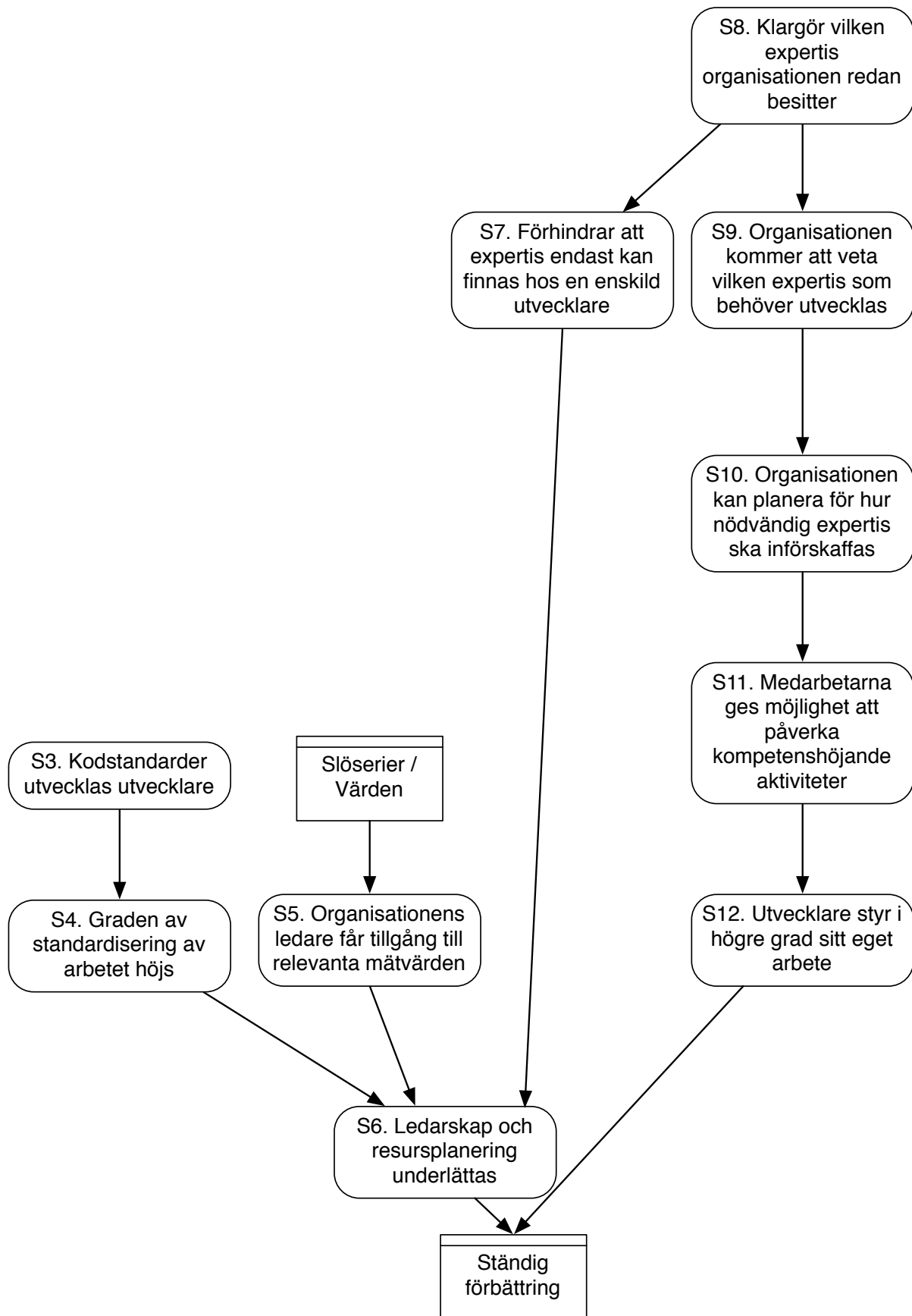
Denna ständiga förbättring kommer till stånd genom att organisationen får möjlighet att kontinuerligt genomföra mätningar, reflektioner och dra lärdom av tidigare arbete.



Figur 9 - Problemggraf avseende ständig förbättring

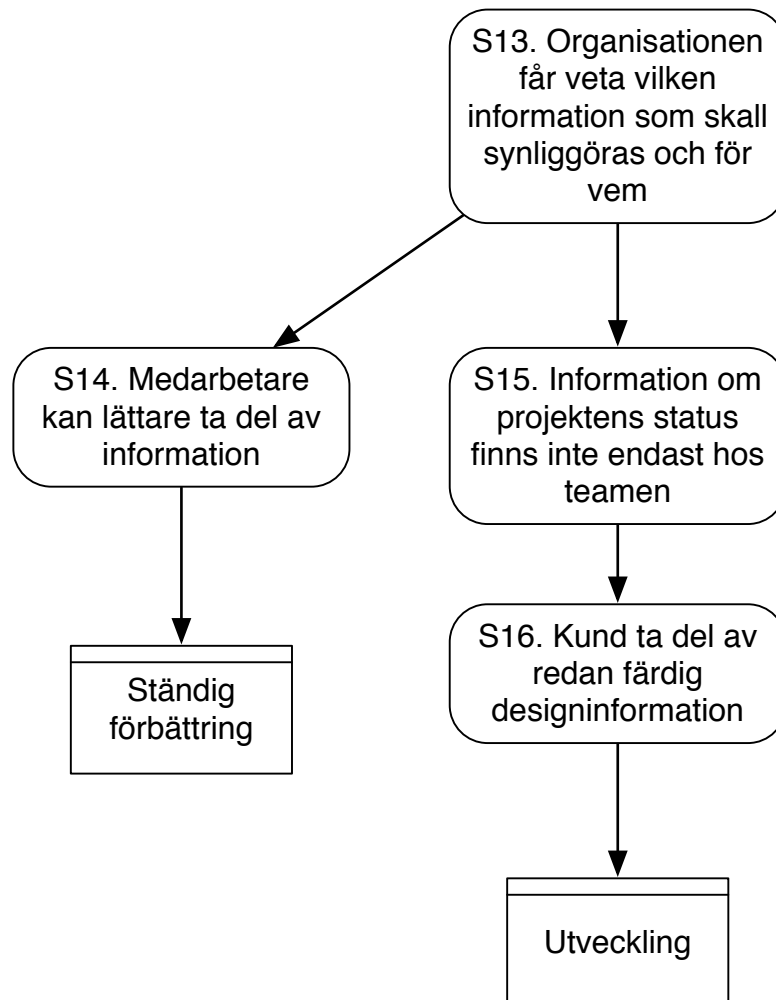
Avsnitt 5.2.2.2 till 5.2.2.5 illustrerar grafiskt relationer som finns mellan de identifierade styrkorna. Notationen beskriver hur en styrka kan möjliggöra eller förstärka andra styrkor.

6.2.2.2 Ledarskap



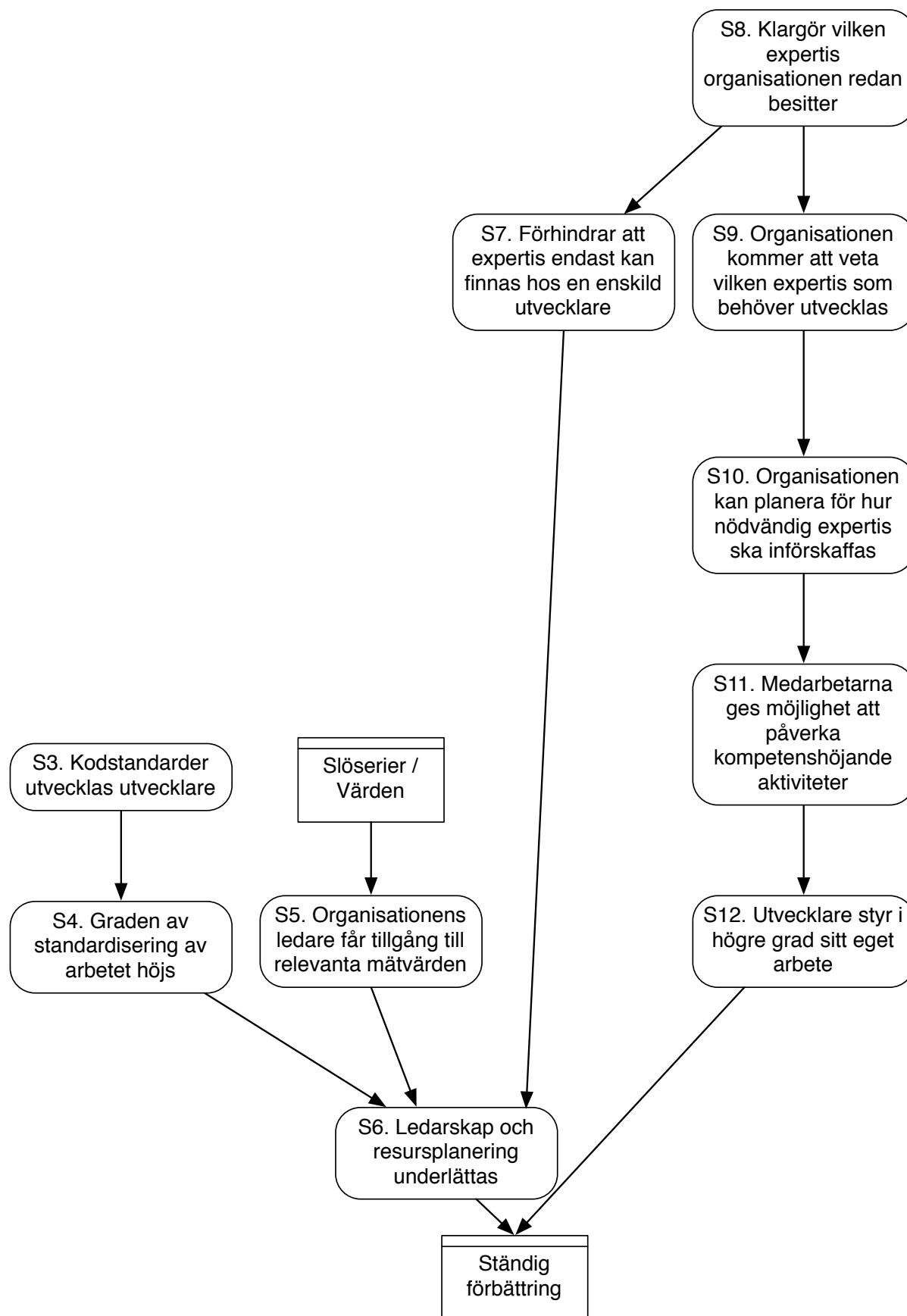
Figur 10- Problemgraf avseende ledarskap

6.2.2.3 Synliggörande av information



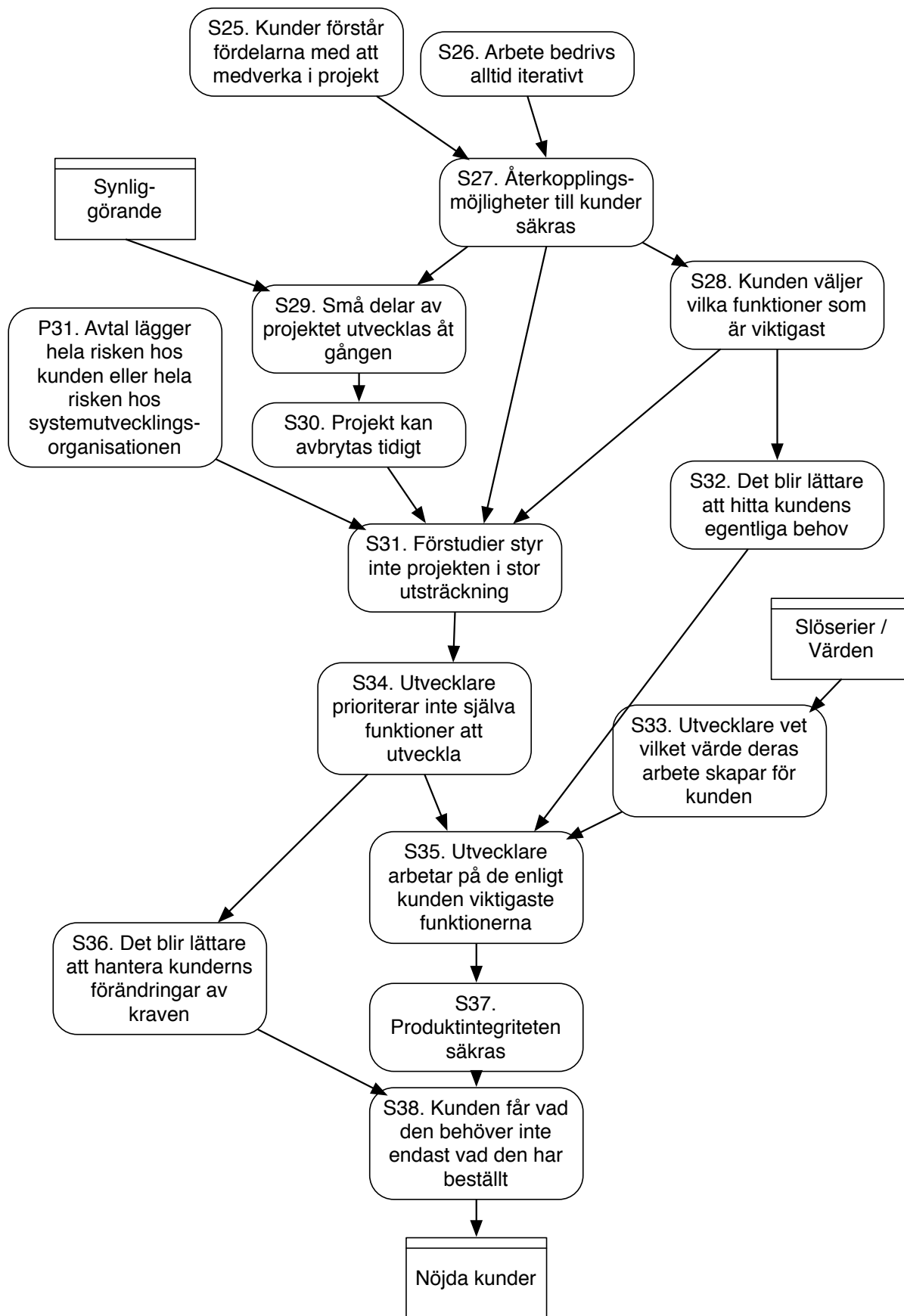
Figur 11- Problemgraf avseende synliggörande av information

6.2.2.4 Slöseri / värden



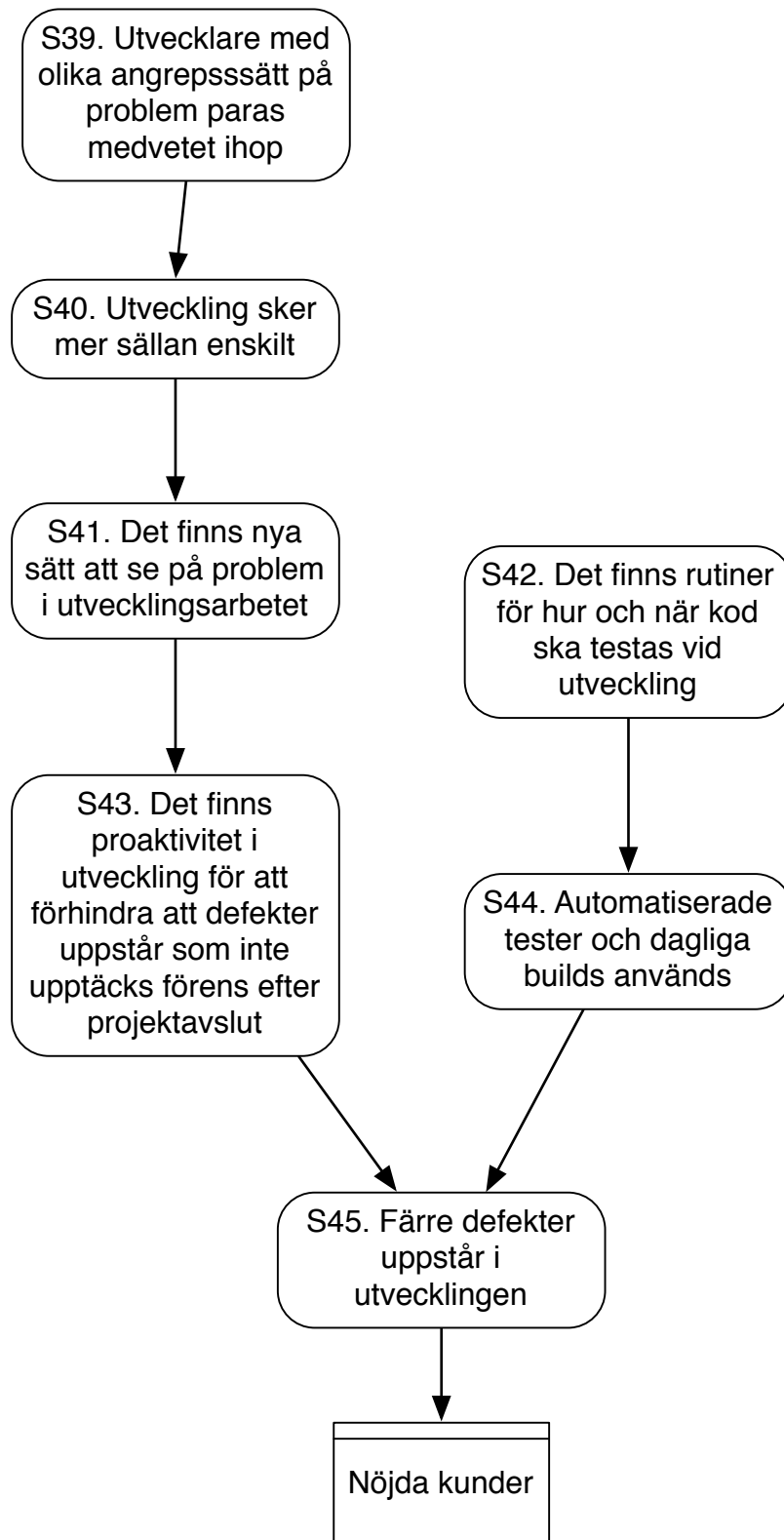
Figur 12- Problemgraf avseende slöserier/värden

6.2.2.5 Utveckling



Figur 13- Problemgraf avseende utveckling

6.2.2.6 Defekter



Figur 14 - Styrkegraf avseende defekter

6.2.3 Generella styrkor med den skapade metoden

Metoden har utformats för att möta de problem som finns gällande beskrivning av Poppendieck & Poppendieck's principer och tankeverktyg. Detta medför att metoden har styrkor som inte sedan tidigare fanns i Poppendieck & Poppendieck's (2003) principer och verktyg. Styrkorna uppkommer genom de mallar som har skapats samt genom att följa den arbetsprocess som beskrivs i kapitel 6.5. Nedan sammanfattas de styrkor som Poppendieck & Poppendieck's (2003) principer och tankeverktyg har och en relation mellan styrkorna och arbetsprocessen/mallarna görs.

I tabellerna benämns arbetsprocessen och mallarna på följande sätt:

Objekt	Förkortning
Arbetsprocessen	AP
Ledarmall	LM
Projektavslutsmall	PM
Iterationsmall	IM
Förstudiemall	FM
tisdag-torsdagmall	TM
Kaizenmall	KM
"6månadersmall"	6M

Styrka	Möts av
Klarlägger vilka organisationens grundläggande värderingar är	6M
Ger organisationen fasta punkter att arbeta med tankeverktygen utifrån	AP
Ger organisationen riktlinjer för dokumentation	LM, PM, IM, FM, TM, KM, 6M
Ger organisationen riktlinjer för vilken slags information som ska synliggöras och för vilka parter	AP

6.2.4 Styrkor med Poppendieck's principer och tankeverktyg

I kapitel 6.2.2 görs en analys av de styrkor som Poppendieck & Poppendieck's (2003) principer och tankeverktyg har. Nedan ges en sammanfattning av styrkorna och hur dessa möjliggörs med hjälp av mallarna och arbetsprocessen.

6.2.4.1 Ledarskap

Styrka	Möjliggörs av
Ger verktyg för att styra mot gemensamma mål	LM, KM, 6M
Beskriver vilken slags information som skall mätas och användas som underlag för ledare	IM, LM, KM
Beskriver hur avtal kan skrivas för att inte lägga risken helt hos en ensam part	LM
Beskriver hur ledare inom organisationen kan arbeta för att hålla medarbetare motiverade	LM, IM, 6M

6.2.4.2 Medarbetare

Styrka	Möjliggörs av
Beskriver hur expertis som behövs ska identifieras och införskaffas	6M
Beskriver hur motivation och bemyndigande av medarbetare ska ske	LM

6.2.4.3 Synliggörande av information

Styrka	Möjliggörs av
Beskriver vilken slags information som bör göras tillgänglig och för vilka intressenter olika slags information ska göras tillgänglig	AP
Visar på vikten av att dokumentation som görs skall vara enkel och att den ska användas för att ha ett existensberättigande	AP, LM, PM, IM, FM, TM, KM, 6M

6.2.4.4 Reflektion

Styrka	Möjliggörs av
Beskriver hur utvecklare kan reflektera kring arbete som har gjorts, ska göras och vilka problem som hindrar fortsatt arbete	IM, FM
Beskriver hur reflektion/diskussion kring kundvärde bör göras	IM, 6M
Beskriver vad som bör diskuteras för att möjliggöra ett ständigt lärande i organisationen	6M, AP
Ger verktyg för att klarlägga/diskutera slöserier inom organisationen	KM

6.2.4.5 Produktintegritet

Styrka	Möjliggörs av
Beskriver hur upplevd och konceptuell integritet kan säkras	IM, LM, FM

6.2.4.6 Arbetsprocess

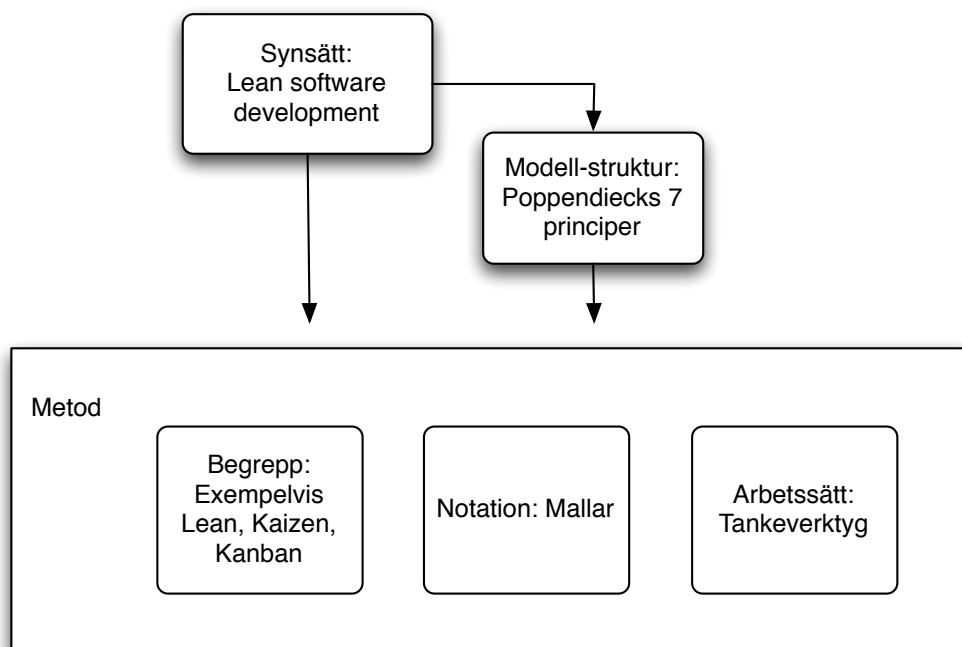
Styrka	Möjliggörs av
Beskriver hur utvecklare kan jobba proaktivt för att undvika defekter	IM, FM
Möjliggör ett pull-baserat arbetssätt vilket minskar kötider	AP
Tydliggör vad syftet med dokumentation av arbetsprocesser är	KM
Ger rutiner för hur utveckling ska bedrivas	IM, FM, TM
Ger verktyg som möjliggör för organisationen att hitta grundläggande värderingar att arbeta utifrån	6M

6.2.4.7 Kunder

Styrka	Möjliggörs av
Beskriver på vilket sätt kunderna ska vara involverade i utvecklingsprocessen	FM
Beskriver vikten av att sätta kunden i centrum	FM, IM, 6M

6.3 Lean-metoden i relation till Goldkuhl's metodstruktur

För att verifiera att den metod som ingår i studiens resultat går att betrakta som en metod har jag valt att validera resultatet mot Goldkuhl's (1991) modell över metodstrukturer. Figur 13 visar hur jag ser på lean-metoden utifrån Goldkuhl's (1991) metodstruktur.



Figur 15 - Metoden beskriven med Goldkuhl's (1991) modell

Lean software development utgör metodens bakomliggande synsätt och vägledande principer. I Poppendieck & Poppendieck's (2003) sju principer som berättar *vad* som bör göras i en lean-metod för att använda *lean production* i systemutvecklingsdomänen, då kallat *lean software development*. Dessa principer utgör modell-strukturen.

Själva metoden innefattar ett antal olika begrepp, såsom lean, kaizen, kanban m.m. För att använda metodens tankeverktyg används den notation som finns i form av mallar (se bilagor A-G).

Lean-metodens sammansättning ligger nära Goldkuhl's (1991) beskrivning av vad som bör ingå i en metod. Inom lean-metoden definieras begrepp i de olika mallar som används för att dokumentera det som utförs med lean-metoden. Dessa mallar styr och beskriver även en stor del av den arbetsprocess som metoden innefattar.

6.4 Metoden LUT - Förslag till lösning på identifierade problem

Avsnittet presenterar en metod för *lean software development* baserad på Poppendieck & Poppendieck's (2003) principer och tankeverktyg vid namn LUT – Lean systemUTveckling.

En beskrivning följer nedan vilken förklarar de delar som finns i LUT's arbetsprocess samt innehållet i dess mallar.

För att visa på vilket sätt delar inom LUT möter de problem som finns gällande användning av Poppendieck & Poppendieck's (2003) principer och tankeverktyg markerar jag löpande i texten vart och hur dessa problem löses. I texten beskrivs LUT som en helhet vilket innebär att den förutom att lösa de generella problemen även möjliggör ett nyttjande av de styrkor som principerna och tankeverktygen för med sig.

För att minimera volymen av dokumentation som cirkulerar på Toyota använder de sig av rapporter som begränsas till ett A3 papper. Genom att tvinga alla att fatta sig kortfattat och kärnfullt blir det enklare att använda rapporter som diskussionsunderlag. Långa studier med sammanfattningar kommer inte att godkännas eftersom ledare inte kommer vilja läsa dessa utan kommer att kräva att få förslagen på ett A3. Det betyder inte att Toyota utelämnar mer information men att de lär sig att skriva kortfattat och kärnfullt. (Liker, 2004)

Jag använde detta synsättet när jag utvecklade de mallar som ingår i LUT. Genom att skapa kortfattade mallar som används i arbetsprocessen nås en större grad av standardisering i organisationen samt möter det problem som Poppendieck & Poppendieck's (2003) tankeverktyg har rörande riktlinjer för dokumentation (GP3, GP5, GP9). Eftersom mallarna hålls kortfattade avhjälpes detta även problemet med att dokumentation som skapas blir för utförlig (GP4). De mallar som LUT använder sig av återfinns i bilaga A-G.

Karaktären på förbättringsförslagen som LUT presenterar innebär både förbättringar och förändringar av strukturen på Poppendieck & Poppendieck's (2003) tankeverktyg.

6.4.1 Ledarskapets uppgifter

Inom *lean software development* har ledarskapet uppgiften att styra organisationens samlade insatser mot gemensamma mål på samma sätt som ledare på Toyota såg till att den bil som producerades liknade tanken som fanns vid starten av processen. För att göra detta använder sig ledare av ett antal olika aktiviteter som inte har någon fast tidpunkt för genomförandet.

Rekommendation är att gå igenom ledarskapsmallen (se bilaga A) varannan månad då den ger en snabb översyn av hur organisationen fungerar.

Nedan följer en beskrivning av det ansvar som organisationens ledare har. Dessa är avsiktligt inte knutna till specifika tidpunkter utan verkställs genom att under verksamhetsåret gå igenom ledarskapsmallen där genomförandet av nedanstående punkter dokumenteras via en checklista.

6.4.1.1 Definiera ett manifest för organisationen

Manifestet ska enkelt presentera de förhållningssätt och värderingar som organisationen skall arbeta efter. Detta skall göras enkelt och så begripligt som möjligt. En gång om året går organisationen igenom manifestet för att se till att det fortfarande speglar de värderingar som organisationen arbetar efter. Initialt behöver ett utkast till manifest skapas av ledarskapet vilket är en engångsaktivitet och inte beskrivs i ledarskapsmallen. Skapandet av detta manifest möter problem GP8.

6.4.1.2 Säkerställ att arbetsbeskrivningar hålls uppdaterade

De arbetsbeskrivningar som finns skall i största mån utgöras av standarder istället för statiska beskrivningar som inte nödvändigtvis speglar det faktiska arbete som utförs. För att undvika detta ska arbetsbeskrivningar utvecklas av de personer som de berör.

6.4.1.3 Kartlägg värdekedjan

Värdekedjan skall kartläggas för att identifiera slöserier och för att klarlägga hur organisationen arbetar. Den diskuteras med resterande delar av organisationen på halvårsvisa möten. Punkten fungerar i mallen som ett extra säkerställande av att denna viktiga aktivitet har utförts.

6.4.1.4 Projektacceptans

Det är ledarskapets uppgift att ge initial acceptans av förfrågan om projekt. Efter att detta har gjorts tillsätts ett team av utvecklare, en eller flera, som kommer att arbeta med projektet. Det är viktigt att detta team har personer som brinner för idén, tillgång till kunder, möjlighet att själva välja sina åtaganden. Inom teamet ska person med breddtänkande ha tillgång till person med viddtänkande.

I denna process är det även ledarskapet säkerställa att projekt har ett tydligt och intressant ändamål som går att nå.

6.4.1.5 Mätningar av arbete

Organisationens arbete ska utvärderas och effektiviseras i syfte att uppnå en ständig förbättring. Det är dock information som skall mätas, inte prestanda. För att upp uppnå detta bör ledare genom att aggregera defektrapporter mäta defekter per projekt, inte vilken enskild utvecklare som har orsakat defekterna.

Mätvärden som används ska motsvara kundens värden. Detta kan göras genom att säkerställa att iterationer dokumenteras och följs upp, kontinuerlig kundkontakt säkerställs och att cykeltider mäts och utvärderas.

6.4.1.6 Stödja utvecklingen

Det är ledarens uppgift att stödja utvecklingen i så stor utsträckning som möjligt. Detta uppnås genom att säkerställa att utvecklare kan komma till ledare med behov som tas på största allvar, det är utvecklarna som utför det dagliga arbetet och genom att säkerställa att deras arbete underlättas skapas produktivitet och effektivitet.

I det dagliga arbetet ska ledare se till att par-programmering möjliggörs där möjligt och att codereviews genomförs i varje iteration för att säkerställa kvalitet på kod som levereras.

6.4.1.7 Kontrakt/Avtal

Ledarskapet måste hitta former som inte lägger hela risken hos kunden eller utvecklingsorganisationen genom att diskutera alternativa avtalsformer med en jurist. Syftet med att hitta andra former är förutom att fördela risken att hitta former som kan bygga tillit mellan parter då återkommande nöjda kunder som upplever ett värde är organisationens främsta resurs.

6.4.1.8 Ledarskapets övriga uppgifter

En viktig uppgift för organisationens ledare är att säkra den långsiktiga riktningen. Ett bra sätt att göra detta är att säkra projekts upplevda integritet genom att matcha språk som kund och utvecklare använder mot språk som används i kod. Genom att säkra hög integritet på levererade produkter säkerställs att kunden sätts i centrum för utvecklingen och att kvaliteten hålls på en hög nivå.

Ledare bör även undersöka problem genom att gå ut i organisationen och se problemet på den plats där det uppstod (Genchi gebutsu) för att få en så sanningsenlig bild som möjligt av situationen. Genom att säkra detta möter LUT även problem GP1.

För att kunna använda LUT i sin helhet bör ledare se till att skapa en organisationskultur som tillåter misslyckanden, håller skeptiker borta från projekt, låter master developers uppstå samt förespråkar och utbildar i ämnet *lean software development*.

6.4.2 Förstudie

Innan ett projekt påbörjas genomförs en förstudie med hjälp av förstudiemallen (se bilaga D). Detta är ett möte där kund och utvecklare tillsammans diskuterar saker för att i så stor grad som möjligt förstå vad det är kunden egentligen vill ha.

Förstudien ska identifiera de punkter som organisationen behöver för att genomföra ett systemutvecklingsprojekt. För att göra detta behöver organisationen veta vad kunden har för behov, vilka krav den har på systemet och systemutvecklingsorganisationen samt vilka tidsramar som organisationen behöver förhålla sig efter.

Utöver detta behöver organisationen veta vad projektet kommer att tillföra kundens verksamhet för värde. Detta är viktigt eftersom metoden betonar vikten av att kunden får det den behöver, inte vad den har beställt. För att uppfylla detta behöver även kunden förstå vikten av att aktivt delta i processen och vad det innebär att aktivt delta.

Prototypbaserad utveckling är ett bra verktyg för att säkra att kunden får det den behöver. Detta innebär att ett antal prototyper tas fram och presenteras för kunden vilka därefter används som diskussionsunderlag. Det är lättare för en kund att snäva in en diskussion om den kan få peka ut det förslag som ligger närmast vad den hade tänkt sig. Prototypbaserad utveckling innebär dock en extra kostnad för kunden och behöver förankras väl genom diskussioner mellan utvecklare och kunden.

Det är även viktigt att tidigt redogöra för vem som har ansvaret för framtida förvaltning och vad en uppskattad kostnad för detta kommer att vara.

Förstudien skall även säkra att utvecklare börjar arbeta på de delar av systemet som är viktigast för kunden. Därför identifieras funktioner som det nya systemet ska tillhandahålla och kunden får efter det prioritera bland dessa funktioner. Efter det utvecklas endast en funktion i taget vilket gör att kunden får det som är viktigast utvecklat först och lätt kan avbryta projektet eller komma med förändringsförslag löpande. Det åligger utvecklaren att se till att organisationen inte åtar sig att utveckla något som den inte har kapacitet att utveckla.

Det är även en rekommendation att en nyttokalkyl skapas där det som utvecklas sätts i relation till vilka förväntade effekter det kan ge för kunden. Om det är möjligt kan en ekonom från organisationen medverka i detta arbete.

6.4.3 Möten som genomförs i systemutvecklingsprocessen

Före och efter iterationer hålls ett antal möten för att säkra att projektet går ett mål som skapar värde för kunden. För att dessa möten används iterationsmallen (se bilaga C).

6.4.3.1 Team-möte med kund före iteration börjar

Möte mellan utvecklingsteam (eller representant för detta vid större team) och kund. Syftet med mötet är att inhämta information från kunden som kan vara relevant för nästa iteration.

För att säkra att det som utvecklas alltid är det som kunden tycker är viktigast genomförs en prioritering av de funktioner som ännu inte utvecklats av kunden. Utvecklare väljer sedan vad iterationen ska innehålla utifrån denna prioriterade lista. Detta eftersom det kan finnas funktioner som rent utvecklingsmässigt bör utvecklas tillsammans. För att kunden bättre ska förstå utvecklingens förväntade tidsåtgång görs en estimering från utvecklare kring den tid som kommer att behövas för att utveckla funktionerna. Efter detta bestämmer utvecklare hur lång tid iterationen kommer att pågå. Det är viktigt att förstå att under en pågående iteration får inte några ändringar från kunden genomföras utan funktionen utvecklas enligt planen för att utvärderas och följas upp på nästkommande möte.

Utvecklare dokumenterar även tänkbara beslut som kan uppstå. Dessa kategoriseras i svåra och lätta beslut och dokumenteras tillsammans med en uppskattning om vilken slags information som kan komma att behövas för att ta beslut inom iterationen.

6.4.3.2 Möten som genomförs efter att en iteration är slut

6.4.3.2.1 Möte med team

Utvecklare dokumenterar efter avslutad iteration två-tre meningar som utgör ett tema för iterationen och kan relaterar iterationen till kundens affärsvärde samt reflekterar kring om de hade det underlag som behövdes för att utveckla det som var av störst betydelse för kunden. I denna process behöver de även dokumentera om de hade tillgång till kunden i den utsträckning som behövdes för att kunna inhämta ytterligare information under iterationen. Allt detta för att säkerställa att projektet håller den höga standard som kunden ska kunna förvänta sig.

För att möjliggöra ett ständigt lärande i organisationen så upprättas en lista med bra och dåliga practices som förekom inom iterationen. Det är även viktigt att diskutera om utvecklarna hade den expertis som behövdes för att utveckla och tidstimerade de funktioner som iterationen bestod av.

Eftersom utvecklare själva behöver bevaka sin arbetssituation behöver de efter varje iteration reflektera över om huruvida de har tillgång till de resurser som behövdes. Det är även viktigt att diskutera om de hade: den frihet, stöd och kunskap som behövdes för att möta iterationens krav, rätt utvecklingsmiljö och om codereviews genomfördes för att säkra kvaliteten i levererad kod.

Teamet behöver även reflektera över hur de spenderade sin tid och om det går det att göra det bättre i framtiden. För att besvara denna fråga behöver de gå djupare och förstå de bakomliggande orsakerna. Detta kan göras genom att fundera över vad som gör arbetet långsamt och vad skulle göra arbetet snabbare/bättre/billigare i framtiden.

Tänkbara problem som kan lösas med refactoring dokumenteras på mötet utifrån områdena enkelhet, klarhet, lämplighet, DRY (Don't repeat yourself) och extra funktioner. Det är viktigt att kunden förstår på vilket sätt dessa påverkar projektets slutresultat och har möjlighet att ta beslut om huruvida de ska åtgärdas eftersom de innebär en extra kostnad för kunden.

På mötet ska de tre längsta köerna identifieras och mål för hur långa dessa bör vara sätts upp. För att nå dessa mål sätts en grupp samman som angriper den längsta kön

6.4.3.2.2 Team-möte med kund

Efter att en iteration är slut genomförs ett möte mellan teamet och kunden. På detta möte dokumenteras och diskuteras det om iterationens resultat löser problemet på det sätt som var meningen. Kunden får här möjlighet att beskriva hur iterationens resultat påverkar vad den behöver. Förändringsbehov som kunden har dokumenteras och en diskussion kring vad detta får för effekter på projektet dokumenteras. Vid detta tillfälle behöver även en diskussion kring vad som behövs för att funktionen ska kunna sättas i produktion genomföras.

För att säkerställa att kunden och teamet har samma uppfattning om resultatet genomförs tester med kund för att studera användbarheten.

Under mötet så diskuteras de refactoringbehov som identifierats under utvecklingen och i det föregående mötet med teamet med kund och teamet får förklara hur åtgärder kring dessa kan leda till ett ökat värde för kunden i förlängningen.

Vid större projekt så genomförs efter var tredje iteration ett möte med en person som inte normalt brukar vara med på dessa möten. Det kan exempelvis vara en användare av det slutgiltiga systemet. Syftet med detta är att brainstorma kring problem med systemet och att skapa en grupp som kan ge sig an de tre viktigaste problemen som uppkommer från denna diskussion. Resultatet från denna grupp följs upp efter två veckor.

6.4.3.3 Efter projektavslut

Utvärdering av genomfört projekt görs efter sista iterationen med hjälp av projektavslutsmallen (se bilaga B).

Dokumentationen skall innehålla bland annat vilken information som behövdes för att ta de mest kritiska besluten under projektet och ska vara baserad på den dokumentation som har skapats innan projektet började samt den dokumentation som har uppstått under iterationerna.

För att hitta lämpliga mätvärden för organisationen ska cykeltider uppskattas.

Detta görs genom att uppskatta den tid det tog från att en funktion kodades till dess att tester kördes, den blev integrerad i systemet, den automatiska testsviten kördes, kundtester/användbarhetstester genomfördes och den satts i produktion.

6.4.3.4 Schemalagda möten

6.4.3.4.1 Tisdag-torsdag varje vecka med teamet

Dessa möten kan genomföras dagligen om det finns tidsutrymme för detta med hjälp av Tisdag-Torsdag-mallen (se bilaga E). Genom att begränsa tidsåtgången för dem till 15 minuter varje tisdag och torsdag kommer en naturlig koppling mellan frågorna att uppstå. På dessa möten dokumenteras snabbt och kortfattat följande frågor.

Vad gjorde jag igår? Vad ska jag göra idag? Vad behöver jag hjälp med?

Alla frågor som kräver en fördjupad diskussion ska tas på möte med den part som är involverad i problemet.

6.4.3.4.2 Måndag varje vecka med avdelningen

Syftet med mötet är att jobba mot kaizen (ständig förbättring). Genom att reflektera över grundläggande saker kopplade till det dagliga arbetet förankras tankeprocessen hos alla medarbetare. Som stöd i detta arbete används kaizenmallen (se bilaga F).

På mötet ska avdelnings-kanban uppdateras. Varje team får i uppgift att uppdatera sitt projekt.

En av de 7 slöserierna (se kapitel 3.3.4) diskuteras varje månad. Syftet med diskussionen är att få medarbetare att reflektera över sitt arbete och om det sätt som det genomförs går att förbättra. Frågor ställs på dessa möten kring huruvida det aktuella slöseriet faktiskt är ett slöseri och vad som kan göras åt det.

Lösningförslag dokumenteras i enkla punkter personer uppmuntras att ta ansvar för att genomföra dessa förslag. Efter detta diskuteras vad som har gjorts för åt slöseriet som diskuterades förra veckan. Denna diskussionsserie bör genomföras en gång per år.

I syfte att sträva mot ständig förbättring så diskuteras den utvecklingsmässigt största utmaningen som finns i organisationen. Tre olika lösningar på problemet tas fram och personer som är villiga att jobba med dessa fram tills nästa möte tilldelas uppgiften. Åtgärderna dokumenteras och följs upp på nästkommande möte.

6.4.3.4.3 Var 6e månad

Mötet är ett tillfälle för att organisationen att reflektera över sitt samlade arbete. På mötet ska följande saker genomföras med hjälp av ”6månadersmallen” (se bilaga G):

6.4.3.4.3.1 Värdeskapande

Organisationens 10 viktigaste aktiviteter sett utifrån kundens ögon betygssätts och därefter planeras hur de lägst rankade problemen skall lösas. Lösningen dokumenteras och följs upp på nästkommande möte. Punkten ska besvara frågor såsom ”Varför finns vi? Varför är det vi gör viktigt? Vad är viktigt för våra kunder?”

Värdekedjan diskuteras utifrån premisser såsom varför den ser ut som den gör och på vilket sätt den kan förändras.

6.4.3.4.3.2 Expertis

Organisationen dokumenterar tillsammans inom vilka områden organisationen besitter expertis och hur den kan utvecklas. En diskussion kring storleken på budgetpost som behövs för att säkra denna kompetensutveckling diskuteras och dokumenteras.

Därefter får alla skriva ner vart organisationen saknar expertis eller områden där expertisen är begränsad till en enskild person. Svaren sammanställs och därefter får alla sätta betyg på de viktigaste områdena för att efter det tillsammans göra en plan för hur denna expertis ska införskaffas inom nästkommande år. Rankningen och lösningen dokumenteras för att följas upp på nästkommande möte.

6.4.3.4.3.3 Underhåll

Utvecklarens upplevelse kring hur ansvaret för att underhålla system som de har utvecklat har fungerat.

6.4.3.4.4 Årligen

Detta är en aktivitet i syfte att stärka sammanhållningen inom organisationen. Diskussioner på detta möte bör ligga på så organisationsövergripande nivå som möjligt.

Organisationens grundläggande värderingar diskuteras och dokumenteras. Frågor såsom varför det som organisationen gör är viktigt diskuteras under detta möte. Uppföljning från föregående möte görs och en diskussion huruvida organisationen fortfarande stödjer det som beskrevs då genomförs. För att genomföra mötet använder organisationen åter igen av "6månadersmallen" (se bilaga G).

Skillnaden mellan detta möte och "6Månadersmötet" är att på detta möte skall en stor del av tiden spenderas till att djupgående diskutera och revidera organisationens manifest.

6.4.4 Synliggör information

En viktig del av LUT är att synliggöra rätt information för rätt part i rätt tid. Nedan beskrivs vilken slags dokumentation som organisationen måste lagra och hur den ska göras tillgänglig för relevanta intressenter. Denna beskrivning möter GP2.

6.4.4.1 För team i projekt

För varje iteration synliggörs information genom att på lämplig plats lagra och tillgängliggöra iterationsmallar och tisdag-torsdag standup rapporter. I dessa återfinns svar på frågor såsom vad som har gjorts, vad som håller på att göras och vad som ska göras.

Utifrån dessa går det att läsa ut klart vilka aktuella problem som finns, förbättringsidéer i form av refactoringkandidater. Organisationen behöver använda sig av system som visar kanban och daglig buildstatus på lämplig placering där människor ofta befinner sig.

6.4.4.2 För organisation

Kanban med status för varje projekt ska finnas tydligt placerad där människor ofta rör sig för att skapa en medvetenhet i organisationen kring vad som händer och vilken status saker har. Det skall även där gå att se vart problem har uppstått och vart extra insatser kan behövas.

Värdekedjan diskuteras på möte var 6:e månad och ska under tiden mellan dessa möten finnas tydligt synliggjord.

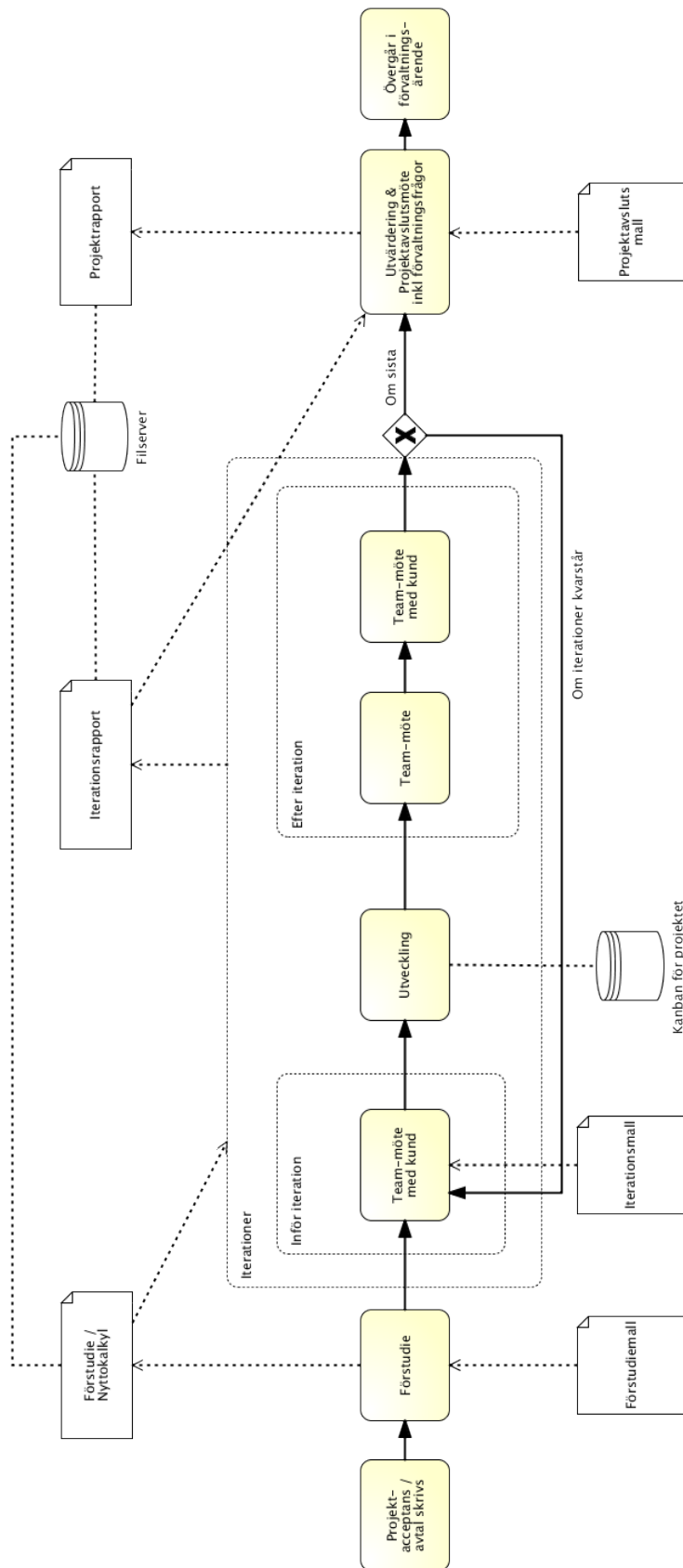
Genom att läsa iterationsmallar och projektrapporter tillgängliggörs information gällande problem. Då i form av förutsedda problem och det uppskattade informationsbehov som behövs för att lösa dessa. Samt i projektrapporter där uppföljning har gjorts för att se vilket informationsbehov som fanns i slutändan och vilka kritiska beslut som behövdes ta.

6.4.4.3 För kund

Kunden skall ha full insyn i dokumentation som rör dess projekt för att lätt kunna följa vad som har gjorts för att nå målet hittills samt vad som återstår att göra för att nå målet. Det är även viktigt att de kan reflektera över systemets affärspåverkan för kunden i dagsläget. Information för att göra detta återfinns i iterationsrapporterna.

Genom att även dela med sig av delvis färdig designinformation samt demo-server under tiden utveckling sker skapas en möjlighet för kunden att göra informerade omprioriteringar vid nästa iterationsmöte. Kund bör lämpligtvis på något sätt dokumentera när en sådan review har gjorts.

6.5 LUT: Arbetsprocess



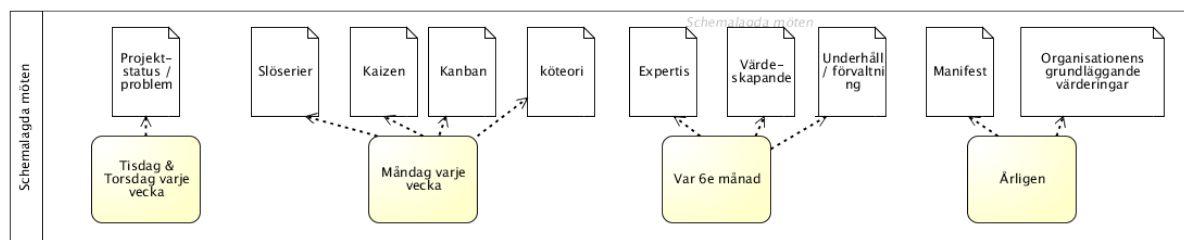
Figur 16 – LUT's arbetsprocess

Figur 12 ger en visuell beskrivning över arbetsprocessens sekventiella utförande. Genom att tydligt strukturera upp när möten med kunder/organisationen skall ske så möter LUT problem GP7, GP6.

Ett projekt inkommer och accepteras av beslutsfattare inom organisationen. Denne tilldelar projektet till ett team av utvecklare som minst består av en enskild utvecklare. Tillsammans med kund genomförs en förstudie baserad på förstudiemallen (Se bilaga D).

När förstudien har accepterats av kunden så påbörjas det iterativa systemutvecklingsarbetet. Före det att en iteration påbörjas genomförs ett möte tillsammans med kund baserat på sida 1 i iterationsmallen (Se bilaga C). Under iteration uppdaterar utvecklare kontinuerligt kanban.

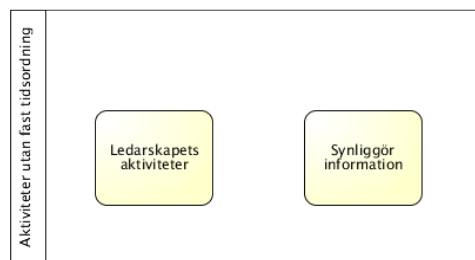
Efter iterationen genomförs först ett möte med teamet baserat på sida 2 i iterationsmallen varpå ett möte med kund genomförs baserat på sida 3 i iterationsmallen (se bilaga C). Processen upprepas till dess att kundens behov har mötts varpå en projektrapport upprättas med hjälp av projektavslutsmallen (se bilaga B). Ärendet övergår därefter till förvaltning.



Figur 17 - Kontinuerliga möten

Parallellt med att arbetet inom arbetsprocessen genomförs så sker kontinuerligt möten med organisationen vilket visas i figur 13. Dessa möten finns beskrivna i mer detalj i kapitel 6.4.

LUT beskriver även ett antal aktiviteter som ledarskapet ska genomföra som inte har någon fast tidpunkt. Dessa bör utföras regelbundet och rekommendationen är att de utförs minst varannan månad för att få effekt. Även synliggörande av information är inte knutet till någon specifik tidpunkt utan är beroende av genomförandet av arbetsprocessens andra aktiviteter. Synliggörande av information beskrivs närmare i kapitel 6.4.



Figur 18 - Aktiviteter utan fasta tidpunkter för utförande

Metoden LUT är skapad för att passa systemutvecklingsorganisationer i allmänhet. Fokus har lagts på att inte knyta aktiviteter eller ansvar till en specifik organisationsstruktur annat än en organisation som har en ledare och en eller flera systemutvecklare anställda. På grund av detta möter LUT även problem GP11. Eftersom en genomgående analys av Poppendieck & Poppendieck's (2003) principer och tankeverktyg har legat som grund för utformningen av LUT kan jag även anse att tankeverktygen med hjälp av LUT närmar sig en fullständighet vilket möte problem GP10.

7 Slutsats

Kapitlet ger en sammanfattande beskrivning av studien och diskuterar övergripande frågor kring tillvägagångssättet och resultatet. Därefter behandlas frågor gällande studiens kunskapsbidrag och generaliserbarhet. Kapitlet avslutas med en utvärdering av studien som helhet och förslag till fortsatt forskning.

7.1 Slutsats

Genom att analysera likheter och skillnader mellan *lean production* och *lean software development* i allmänhet har jag verifierat att de ligger varandra mycket nära och faktiskt kan anses ha samma principiella grund, men i två olika tillämpningsdomäner. Detta styrks genom den jämförelse mellan *lean software development* och *lean production* som har gjorts med hjälp av Liker's (2004) principer för *lean production* (se avsnitt 6.1). Eftersom de ligger varandra så nära har jag konstaterat att de således även rimligtvis innehar samma styrkor och problem.

Denna slutsats möjliggjorde att styrkor och problem kring *lean software development* kunde identifieras genom att analysera Poppendieck & Poppendieck's (2003) principer och tankeverktyg. Analysen har även varit möjlig eftersom Poppendieck & Poppendieck's (2003) principer utgör en konkretisering av *lean software development*. (se 3.3.1).

Analysen har lett till att jag har konstaterat att det finns en förbättringspotential i Poppendieck & Poppendieck's (2003) principer och tankeverktyg (se avsnitt 6.2).

Förbättringspotentialen har identifierats genom diskussioner kring principerna och verktygen tillsammans med praktiker (se bilaga H). Utifrån det underlaget har jag utvecklat en konkret metod, LUT (Lean Utveckling), för *lean software development* baserat på de generella problemen som identifierats kring principerna och verktygen. Ett sätt att konkretisera LUT har varit att utveckla mallar som beskriver hur dokumentation ska göras för att säkra kontinuerligt lärande och en genom en arbetsprocess som beskriver de aktiviteter och händelser som möjliggör ett arbete med *lean software development*.

LUT utvecklades tillsammans med praktiker och varje iteration gjorde LUT's struktur tydligare när den prövades mot verksamheten i form av föreståndarens uppfattning av situationen. Detta för att se hur den skulle passa i det sätt som de redan arbetar.

Poppendieck & Poppendieck's (2003) beskriver i sina verktyg inte i detalj den övergripande processen att ta fram en förstudie, kunder har dock ett behov av tidsramar för projektet för att kunna motivera detta i sin egen organisation. Små kompletterande anvisningar kring hur detta ska gå till har därför gjorts i LUT. Genom att LUT har skapats tillsammans med praktiker kan saker som dessa identifieras och inkorporeras för att stärka resultatets användbarhet för praktiker.

Under konversationer med föreståndaren för InnovationLab har en komplett modellering av nuvarande arbetssituation genomförts, det största bidraget i att göra denna har varit att det har

skapat en djup förståelse hos både mig och föreståndaren för hur det vi diskuterat i konversationer faktiskt kan relateras till organisationen⁶.

Detta gör även studien lämplig att använda som underlag för vidare forskning då den är skapad inom ramarna för akademien.

7.2 Utvärdering

7.2.1 Validitet i resultatet

Resultatet har validerats på flera sätt. Dels genom djupgående konversationer med praktiker och dels genom teoretisk validering.

Metoden LUT kommer gå att använda i en systemutvecklingsorganisation eftersom praktiker har varit involverade hela processen från identifikation av problem fram till färdigställandet av metoden.

Teoretiskt har metodens underlag säkrats genom att verifiera att principerna som ligger till grund för *lean software development* är desamma som de inom *lean production*. Metodens giltighet som metod har säkrats genom att ställa den emot Goldkuhl's modell för vad en metod skall innehålla. Resultatet av denna validering visar att metoden i alla avseenden innehåller de delar som utgör en metod.

7.2.2 Validitet i datainsamling

Jag har i datainsamlingen identifierat företeelser som är relevanta för studien. Dessa har tolkats och jag har förstått hur dessa fungerar inom ramen för en systemutvecklingsorganisation. Genom att datainsamlingen har skett i så omfattande former säkras det att jag har skaffat ett så pass komplett underlag som behövs för att göra en korrekt tolkning av föreståndarens världsbild.

Det bör dock påpekas att datainsamlingen brister till viss del då konversationerna inte har spelats in. Att göra detta skulle ha medfört såväl positiva som negativa effekter på datainsamlingen. Genom att spela in konversationerna skulle jag ha fått en möjlighet att gå tillbaks och säkra saker som tonfall, tveksamheter eller betoningar. Att inte spela in dem medförde att jag fick tillgång till mindre filtrerad information och att föreståndaren kunde tala fritt utan att behöva fundera över hur det skulle kunna tolkas. Genom att föra minnesanteckningar och att tillsammans skapa dokument och modeller känner jag dock att det material som samlades in håller en hög validitet. Genom att analysera materialet som skapades (se bilaga I) går det att relatera de inre logiska delarna till en meningsfull helhet.

7.2.3 Validitet i dataanalys

Insamlad data från de informella konversationerna har analyserats genom att studera föreståndarens avvikande meningar från det diskuterade underlaget. Då föreståndaren har haft så få avvikande meningar har det även inneburit att dataanalysen direkt har kunnat inkorporeras i resultatet. Jag har dock lagt stor vikt vid att inte beakta avvikelser som jag har ansett inte kan generalisera till fler systemutvecklingsorganisationer samt har valt att ej behandla specifika verktyg som ligger utanför studiens ramar.

⁶ Modellering av nuvarande arbetsprocess genomfördes under 2 möten 2011-03-18 och 2011-03-28. Denna har inte använts som underlag vid skapandet av metoden annat än som referensmaterial för att se vilka aktiviteter organisationen i dagsläget behandlar.

7.2.4 Kommunikativ validitet

Minnesanteckningar och skapade dokument möjliggör för läsaren att själv bilda sig en uppfattning kring studiens relevans och validitet. Det är möjligt för en läsare att utifrån studiens innehåll se hur resultatet förhåller sig till empirin.

7.2.5 Generaliserbarhet

Trots att resultatet bygger på studier av en enskild systemutvecklingsorganisation så anser jag att resultaten är användbart av andra systemutvecklingsorganisationer med liknande förutsättningar då jag trots den djupa praktikförankringen har fokuserat på att göra studien så organisationsoberoende som möjligt.

Dock så har ett försök att göra metoden mer användbar för praktiker inneburit att LUT har anpassats efter den befintliga mötesstruktur som den studerade organisationen har i dagsläget. Detta påverkar inte metodens validitet men visar att den studerade organisationen kan ha haft stort inflytande på LUT's utformning även när det inte har varit helt uppenbart för mig att det har varit så. Jag anser dock att detta inte är något som påverkar generaliserbarheten då strukturen inte har några beroenden som gör att den inte kan användas av andra organisationer.

Jag tror att LUT kommer att kunna användas av praktiker direkt eller fungera som ett underlag för diskussion och vidareutveckling av befintliga systemutvecklingsprocesser.

Eftersom jag har valt att förlita mig på endast en kvalitativ källa uppstår en brist i generaliserbarhet. Jag anser dock att studien överlag har en god generaliserbarhet som kan ökas allteftersom fler studier genomförs på ämnet.

7.2.6 Kongruens

Studien följer en röd tråd genomgående där varje påföljande del går att relatera till de delar som ligger före. Det är möjligt för läsaren att se att studiens olika delar har vävts samman till en sammanhållen argumentation.

7.2.7 Relation till annan kunskap

Studien har identifierat de viktigaste tidigaste kunskapsbidragen inom området och skapar utifrån dessa ny kunskap. Dessa redogörs för i kapitel 3.

7.2.8 Metodutvärdering

Detta avsnitt syftar till att analysera de metoder som jag har valt att använda mig av i kapitel 2 – Vetenskapligt förhållningssätt och metod.

Min förkunskap har genomgående betraktats som en tillgång i studien och har kontinuerligt utvecklats under studiens gång. Jag har således inte kunnat inta en objektiv forskarroll utan har låtit mina egna tankar och tolkningar få stort inflytande över studiens resultat. Genom att arbeta tillsammans med praktiker har jag dock sett till att förankra mina tankar i verkligheten.

Studien har använt sig av det hermeneutiska perspektivet genom att tolka och analysera språk vid de informella konversationer som har genomförts tillsammans med föreståndaren för den studerade organisationen. Genom att koppla delarna från dessa samtal till en helhet i form av resultatet kan jag även hävda att ett holistiskt perspektiv har anlagts på forskningen. Resultatet är inte meningsfullt förens dess del sätts samman till en helhet.

Då datainsamling har gjorts genom att studera och tolka språket i teori och konversationer går det att påvisa att det är kvalitativ forskning som har utförts. Att den insamlade data som studien behandlar inte går att mäta utan är en unik kombination av egenskaper och kvalitéer styrker detta påstående.

Genom att kontinuerligt generera ny data vid informella konversationer med föreståndaren och utifrån dessa skapa ny kunskap som sedan valideras i ytterligare konversation ligger nära den hermeneutiska spiralen. Det hade varit möjligt att fortsätta detta arbete en längre tid för att skapa ett bättre resultat och för att ytterligare validera detta.

Studiens litteraturstudie genererade dels en sammanställning kring tidigare forskning men gav även ett arbetsmaterial i form av sammanställningen av Poppendieck & Poppendieck's (2003) principer och tankeverktyg. Dessa har djupgående studerats tillsammans med föreståndaren men har även använts för att identifiera problem och styrkor med *lean software development* och *lean production*. Jag har även använt litteraturstudien för att studera hur väl principerna inom *lean software development* och *lean production* stämmer överens.

Efter detta anlades vid metodskapandet ett explorativt arbete med att skapa metoden LUT. Under denna process så identifierades ytterligare problem och styrkor med Poppendieck & Poppendieck's (2003) principer och tankeverktyg. Även dessa har diskuterats vid informella konversationer med föreståndaren. Processen har även varit av en metodutvecklande typ och har fungerat väl för att besvara studiens forskningsfrågor.

Beskrivningen av arbetssättet ovan visar hur det abduktiva arbetssättet har använts för att utifrån teorin deduktivt generera ny kunskap vilken har prövats mot verkligheten för att därefter användas induktivt som underlag för att skapa ny teori i form av metoden LUT.

Det urval som har gjorts har varit väl lämpat för studien främst på grund av tillgängligheten till föreståndaren, hade ett annat urval gjorts hade åtkomsten med största sannolikhet varit betydligt sämre. Det är dock värt att notera att om ett bredare urval med fler respondenter som arbetar med *lean software development* hade gjorts så är det möjligt att studien hade fått ett annorlunda resultat. Jag anser det dock som riktigt att ha valt en organisation som i dagsläget inte arbetar med *lean software development* för att kunna skapa en metod som inte kräver tidigare kunskap inom ämnet från utförarna. Det urval som har gjorts av litteratur skulle ha kunnat systematiseras tydligare men jag anser att det urval som studien har använt sig av väl täcker in den viktigaste tidigare forskningen genom att utgå från antal citeringar.

Insamling av data har skett genom informella konversationer vilka inte har spelats in eller transkriberats. Detta är en brist i arbetet men eftersom jag syftade till att komma åt ofiltrerad och utförlig data har den tjänat mitt syfte väl. Delar av det som diskuterades har även skett genom att exempelvis tillsammans modellera och utvärdera text vilket inte hade gått att spela in. Vid dessa konversationer har olika teman diskuterats fritt och jag har låtit föreståndaren själv styra samtalet i stor utsträckning. Att det inte går att gå tillbaks till materialet för att verifiera en uppfattning är dock en svaghet vilket gör att min egen uppfattning får en extra stor påverkan på studien.

Analysen har genomförts på två olika sätt. Först har en analys av principerna kring *lean software development* och *lean production* gjorts genom att ställa Liker (2004) mot Poppendieck & Poppendieck (2003) detta var ett bra sätt att besvara den första delfrågan och möjliggjorde för mig att besvara den andra delfrågan.

Den andra delfrågan har besvarats genom en styrke- och problemanalys av Poppendieck & Poppendieck's (2003) principer och tankeverktyg. Denna har validerats och kompletterats vid informella konversationer med föreståndaren.

7.3 Förslag på fortsatt forskning

En del av det som dokumenteras skulle i framtiden möjligtvis gå att standardisera och utveckla. Det kan exempelvis vara en idé att ta fram standardiserade motiveringar för hur en ökad klarhet i koden leder till effektivare arbete eller minskade upprepningar leder till enklare underhåll.

Studien behandlar inte i detalj vilka slags verktyg som skall användas för att möjliggöra framförallt synliggörande av information. Det finns här möjlighet att studera hur ett automatiserat sätt att göra detta skulle kunna se ut i praktiken, exempelvis med hjälp av ett elektroniskt kanbanverktyg. Tidigare studier kring användandet av elektronisk kanban har endast varit hypotetiskt upplagt och teoretiskt prövat (Wan & Chen, 2007).

Dokumentationen är nu begränsad till de specifika aktiviteter och händelser som LUT beskriver. Det finns ett utrymme att studera vilken annan slags dokumentation som skulle behövas inom organisationen utöver det som behandlas av LUT och vilket värde detta kan skapa. Exempelvis så betraktar jag nu information som uppkommer från tester och genom kommentarer i kod som dokumentation som uppkommer ”av en slump” och inte hanteras på ett standardiserat sätt i syfte att möjliggöra en ständig förbättring.

En brist är att jag inte har prövat metoden i verkligheten. Metoden bör prövas och värderas av en framtida studie. Eftersom metoden inte har prövats i verkligheten kan den nu endast ses som en välgrundad hypotes.

8 Litteraturförteckning

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J., (2002). Agile software development methods. Review and analysis. *Vtt Publications*, vol 478, ss. 107

Andersen, E.S., (1994a) *Systemutveckling - principer, metoder och tekniker*. Lund: Studentlitteratur.

Andersen, H., (1994b). *Vetenskapsteori och metodlära - Introduktion*. Lund: Studentlitteratur.

Anderson, J.D., (2004). *Agile management for software engineering: Applying the theory of constraints for business results*. Laflin, Pennsylvania, USA: Pearson Education.

Brinkkemper, S., (1996). Method engineering: engineering of information systems development methods and tools. *Information and software technology*, vol 38(4), ss.275-280.

Conboy, K. & Duarte, V., (2010). Scaling Agile to Lean - Track Summary. *Lean Enterprise Software and Systems*, vol 65, ss.1-2.

Cook, J. & Semouchchak, V., (2004). Lean Object-Oriented Software Development. *SAM Advanced Management Journal*, vol 69(2), ss.12-22.

Cronholm, S., (1995). Why CASE Tools in Information Systems Development?-an empirical study concerning motives for investing in CASE tools., *Proceedings of the 18th Information Systems Research Seminar in Scandinavia: IRIS.*, Gjern, Danmark.

Cronholm, S., (2008). Using Agile Methods ? – expected effects., *the 17th International Conference on Information Systems Development (ISD2008)*. Paphos.

Egidius, H., (1986). *Positivism – Fenomenologi – Hermeneutik*. Lund: Studentlitteratur.

Goldkuhl, G., (1991). Stöd och struktur i systemutvecklingsprocessen., *Systemutveckling i praktisk belysning*. Norrköping.

Goldkuhl, G., (1994). *Metodanalys: En beskrivning av metametoden SIMM*. Linköping: Linköpings universitet

Goldkuhl, G., (1998). *Kunskapande*. Linköping: Linköpings universitet

Goldkuhl, G., (2011). *Kunskapande*. Linköping: Linköpings universitet.

Goldkuhl, G. & Röstlinger, A., (1988). *Förändringsanalys - Arbetsmetodik och förhållningssätt för goda förändringsbeslut*. Lund: Studentlitteratur AB.

Högskolan i Borås, IDA, Informatik. (2003). Uppsatskrav. (Elektronisk). Tillgänglig: <[http://www.hb.se/wps/wcm/connect/?MOD=PDMProxy&TYPE=personalization&ID=NONE&KEY=NONE&LIBRARY=%2FcontentRoot%2Ficm%3Alibraries\[7\]%2FIDA%2FDokument%2FStudent%2FExamensarbete&DOC_NAME=%2FcontentRoot%2Ficm%3Alibrarie](http://www.hb.se/wps/wcm/connect/?MOD=PDMProxy&TYPE=personalization&ID=NONE&KEY=NONE&LIBRARY=%2FcontentRoot%2Ficm%3Alibraries[7]%2FIDA%2FDokument%2FStudent%2FExamensarbete&DOC_NAME=%2FcontentRoot%2Ficm%3Alibrarie)>

s[7]%2FIDA%2FDokument%2FStudent%2FExamensarbete%2FUppsatskravinformatik.pdf
&VE 73
RSION_NAME=NONE&VERSION_DATE=NONE&IGNORE_CACHE=false&CONVER
T=text/html&MUST_CONVERT=false> (2011-06-18)

Jayaratna, N., (1999). *Understanding and evaluating methodologies*. Berkshire, England: McGraw-Hill.

Nilsson, J., Lagerstedt, A., Nilsson, J., (2009). *Hur väl fungerar agila metoder i praktiken? - En studie av metoden scrum*. Kandidatuppsats. Borås: Högskolan i Borås Institutionen för data och affärsvetenskap.

Karlsson, D., (2008). *Förändrad syn på arbetsprocesser genom tillämpning av Lean*. Kandidatuppsats. Borås: Högskolan i Borås Institutionen för data och affärsvetenskap.

Liker, J.K., (2004). *The Toyota Way - 14 Management Principles from the World's Greatest Manufacturer*. New York, USA: McGraw-Hill.

Lind, M., Appelqvist, R., Forsgren, O. & Rihs, A.-S., (2006). *Etablering av utvecklingscenter för realisering av IT-användning*. PM. Högskolan i Borås.

Middleton, P., (2001). Lean Software Development : Two Case Studies. *Software Quality Journal*, vol 01 December, ss.241-52.

Morgan, G., (1943). *Images of organization*. USA: Saage Publications, Inc.

Morgan, T., (1998). *Lean Manufacturing Techniques Applied to Software Development*. Masteruppsats. Cambridge: Massachusetts Institute of Technology.

Oates, B.J., (2006). *Researching Information Systems and Computing*. London: SAGE Publications Ltd.

Patel, R. & Davidson, B., (2003). *Forskningsmetodikens grunder - Att planera, genomföra och rapportera en undersökning*. Lund: Studentlitteratur.

Patton, M.Q., (2002). *Qualitative research and evaluation methods*. 3rd ed. Thousand Oaks, Kalifornien, USA: Sage Publications, Inc.

Poppendieck, M. & Poppendieck, T., (2003). *Lean software development: an agile toolkit*. Crawfordsville, Indiana, USA: Addison-Wesley.

Røvik, K.A., (1998). *Moderna organisationer - trender inom organisationstänkandet vid millennieskiftet*. Malmö: Fagbokförlaget.

Raman, S., (1998). Lean software development: is it feasible? In *Digital Avionics Systems Conference*, 1998. 17th DASC. The AIAA/IEEE/SAE.

Thane, M., (1998). *Lean manufacturing techniques applied to software development*. MSc uppsats. Massachusetts: Massachusetts Institute of Technology System Design and

Management Program. (Morgan, T. 1998. Lean Manufacturing Techniques Applied to Software Development, MSc Thesis, Massachusetts Institute of Technology.

Wan, H.-d. & Chen, F.F., (2007). A Web-based Kanban system for job dispatching, tracking, and performance monitoring. *The International Journal of Advanced Manufacturing Technology*, 38(9-10), pp.995-1005.

Womack, J.P., Jones, D.T. & Roos, D., (1990). *The machine that changed the world*. New York: Rawson Associates.

9 Bilagor

Bilaga A - Ledarmall

Ledarskapsmall			
Datum:	Plats:	Utfärdare:	
Kategori	Fråga	Åtgärd	OK
Arbetsbeskrivningar	Har arbetsbeskrivningar upprättats?		<input type="checkbox"/>
Värdekedjan	Har värdekedjan kartlagts?		<input type="checkbox"/>
	Har värdekedjan diskuterats?		<input type="checkbox"/>
	Har alla söserier diskuterats?		<input type="checkbox"/>
	Har uppföljning gjorts?		<input type="checkbox"/>
Projektaceptans	Har accepterade projekt:		
	Haft ett intressant och tydligt ändamål?		<input type="checkbox"/>
	Haft team med personer som brinner för idén?		<input type="checkbox"/>
	Haft team med tillgång till kunder?		<input type="checkbox"/>
	Haft team som kunnat välja sina egna åtaganden?		<input type="checkbox"/>
	Kombinerat breddtänkande med viddtänkande?		<input type="checkbox"/>
Mätning	Har defekter mätts per projekt?		<input type="checkbox"/>
	Har det säkrats att ansvarig för defekter inte sparats utan endast defekter?		<input type="checkbox"/>
	Har iterationer dokumenterats för alla projekt?		<input type="checkbox"/>
	Har uppföljning av dokumentation från iterationer gjorts vid projektavslut?		<input type="checkbox"/>
	Har kontinuerlig kundkontakt säkrats inom projekt?		<input type="checkbox"/>
	Motsvarar de mätvärden som använts kundens värden?		<input type="checkbox"/>
	Har cykeltider mätts?		<input type="checkbox"/>
	Har cykeltider utvärderats?		<input type="checkbox"/>
Stödja utvecklingen	Har utvecklare inkommit med behov?		<input type="checkbox"/>
	Har behoven tagits på allvar?		<input type="checkbox"/>
	Har par-programmering underlättats under perioden?		<input type="checkbox"/>
	Har codereviews genomförts inom iterationer?		<input type="checkbox"/>
Avtal	Har avtal upprättats som inte lägger hela risken hos organisationen eller hos kund?		<input type="checkbox"/>
Övrigt	Har arbetet styrts mot ett långsiktigt mål?		<input type="checkbox"/>
	Har den upplevda integriteten i levererade produkter säkrats?		<input type="checkbox"/>
	Har problem studerats på den plats där de uppstod?		<input type="checkbox"/>
Kultur	Tillåts misslyckanden?		<input type="checkbox"/>
	Har skeptiker hållits borta från projekt?		<input type="checkbox"/>
	Tillåts master developers att uppstå?		<input type="checkbox"/>
	Har <i>lean software development</i> kontinuerligt förespråkats?		<input type="checkbox"/>

Bilaga B - Projektavslutsmall

Projektavslutsmall				
Datum:	Plats:	Utfördare:	Projekt:	
<i>Funktioner utvecklade</i>		Inom iteration nr:	Tidsestimat:	Verklig tid:
<i>Svåra beslut som togs</i>		<i>Uppföljning</i>		
		Planerad start-slut		
		Verkligt start-slut		
		Checklista		
		Cykeltider uppskattade utifrån Iterationsdokumentation och kanban?		<input type="checkbox"/>
		Mål satta för cykeltider?		<input type="checkbox"/>
		SLA upprättat?		<input type="checkbox"/>
		Iterationsdokumentation genomläst?		<input type="checkbox"/>
		Kanban uppdaterad?		<input type="checkbox"/>
		Uppskattad tid från dess att funktion har kodats till dess att:		
		<i>Tester har körts</i>		
		<i>Integrerad i systemet och automatiskt testsvit har körts</i>		
		<i>Kundtester genomförts</i>		
		<i>Användbarhetstester körts</i>		
		<i>Den är satt i produktion</i>		
		Tid		

Bilaga E - Tisdag-torsdagmall

Tisdag-torsdag-mall			
Datum:	Plats:	Utfärdare:	
Tisdag <i>Vad gjorde jag igår?</i>	Vecka -	Torsdag <i>Vad gjorde jag igår?</i>	Svar
<i>Vad ska jag göra idag?</i>		<i>Vad ska jag göra idag?</i>	
<i>Vad behöver jag hjälp med?</i>		<i>Vad behöver jag hjälp med?</i>	
Tisdag <i>Vad gjorde jag igår?</i>	Vecka -	Torsdag <i>Vad gjorde jag igår?</i>	Svar
<i>Vad ska jag göra idag?</i>		<i>Vad ska jag göra idag?</i>	
<i>Vad behöver jag hjälp med?</i>		<i>Vad behöver jag hjälp med?</i>	

Bilaga G - "6Månadersmall"

6månadersmall			
Datum:	Plats:	Utfärdare:	
Organisationens tio viktigaste aktiviteter sett från kundens perspektiv			
Beskrivning	Org. Betyg	Fråga	Svar
1.		Varför finns vi?	
2.		Varför är det vi gör viktigt?	
3.		Vad är viktigt för våra kunder?	
4.		Sammanställning kring gruppövning: Vilka områden har vi stark expertis inom?	
5.			
6.			
7.			
8.			
9.			
10.			
Åtgärder för att förbättra de tre högst rankade			
1.		Sammanställning kring gruppövning: Vilka områden behöver vi skaffa expertis inom?	
2.			
3.			
Checklista		Hur stor budgetpost behövs för att införskaffa denna expertis?	kr
Organisationens manifest reviderat?		Hur fungerar svaret med att underhålla egna utvecklade system?	
		<input checked="" type="checkbox"/>	<input type="checkbox"/>

Bilaga H - Diskussionsunderlag/resultat

Nedanstående är resultatet av de informella samtal och gemensamma arbete som har utförts tillsammans med den studerade organisationens föreståndare. Mina tankar har markerats med (Björklund) medan Poppendieck & Poppendieck's förslag till handling är markerade med (Poppendieck)

Princip	Poppendieck verktyg
Eliminera slöseri	Sju typer av slöseri Mappning av värdekedjan
Förstärk lärandet	Återkoppling Iterationer Synkronisering Set-baserad utveckling
Ta beslut så sent som möjligt	Optioner "The last responsible moment" Djup först vs. Vid först vid problemlösning
Leverera så snabbt som möjligt	Pull system Köteori Förseningens kostnad
Bemyndiga teamet	Självbestämmande Motivation Ledarskap Expertis
Bygg in integritet	Upplagd integritet Konceptuell integritet Refactoring Testning
Se helheten	Mätningar Kontrakt

Verktyg	Förslag till åtgärd	Föreståndares reflektioner
De sju slöserierna	<p>Möte var 6e månad (Björklund) där de 10 viktigaste aktiviteterna i organisationen utifrån kundens perspektiv betygssätts utifrån hur högt eller lågt kunden värderar aktiviteten. Planera hur tidsåtgången för dessa aktiviteter ska halveras. (Poppendieck)</p> <p>Diskutera varje slöseri var för sig på de sju nästkommande veckomötena och enas om det är ett slöseri. Uppskatta hur lång tid slöseriet tar varje vecka. Diskutera vad som kan göras för att minimera detta slöseri. (Poppendieck)</p>	<p>1. Bra – Frågan vi skall ställa oss är ungefär: vad håller vi på med och varför är detta viktigt för kunden. Varför finns vi? I samband med detta kan man tänka sig att vi diskuterar värden för kund.</p> <p>Ex: Reflektioner är viktigt för att vi bidrar till forskare</p> <p>2. Slöserimöte är bra – frågan är hur lång tid</p>

	Diskutera vad som har gjorts föregående vecka för att minimera slöseriet som diskuterades sist (Björklund)	man skall lägga på detta. Hur skall det protokollföras och följas upp? Excel/kanban?
Mappning av värde-kedjan	<p>Mappa värdekedjan (Poppendieck, 2003) och gör denna tillgänglig för organisationen (Björklund)</p> <p>Diskutera den mappade värdekedjan på ett måndagsmöte (Björklund)</p>	<p>Absolut – men följ upp denna en gång i halvåret – kanske vid samma möte som ovan? Finns det nya/förändrade/borttagna värden och stödjer vi fortfarande befintliga... Måndagsmöte – nja...första gången behövs mer tid – uppföljning mindre... Skapa ett InnovationLabmanifest – coolt! En blandning av lean, xp, scrum, forskning etc</p>
Återkoppling	<p>Dokumentera i arbetsbeskrivning för utvecklare följande (Björklund)</p> <ol style="list-style-type: none"> 1. Testa direkt när kod skrivs (Poppendieck, 2003) 2. Pröva idéer i kod och verktyg i praktiken istället för att dokumentera dem (Poppendieck, 2003) <p>Skapa ett dokument för organisationen som beskriver hur den arbetar (Björklund)</p> <ol style="list-style-type: none"> 1. Allt arbete ska göras direkt mot en kund (Poppendieck, 2003) 2. Återkopplingsmöjlighet till kund skall alltid finnas (Poppendieck, 2003) <p>Öka antalet återkopplingar (Poppendieck, 2003)</p>	<p>1 – Automatiserade tester är bra, passar det i alla projekt? Hur gör vi med projekt som redan är utvecklade eller håller på att utvecklas?</p> <p>Kunden måste ha tid att vara med i projektet. Ibland vill inte kunden ta sig denna tiden.</p> <p>3. Ja</p>

<p>Iterationer</p>	<p>Använd en enkel förstudiemall som kan gås igenom tillsammans med kunden (Björklund)</p> <ol style="list-style-type: none"> 1. Kunden beskriver vilka funktioner som behövs, utvecklare hur lång tid de kommer att ta. (Poppendieck, 2003) 2. Utvecklare ska ansvara för att antalet funktioner håller sig på en nivå som organisationen kan klara av (Poppendieck) <p>Iterationer kan ta olika lång tid men en maxtid för iterationer sätts inför varje projekt. (Björklund) (Poppendieck säger att det finns olika syn på det hela)</p> <p>Möte med kund inför varje iteration där följande diskuteras:</p> <ol style="list-style-type: none"> 1. Kund prioriterar funktioner 2. Utvecklare väljer funktioner från toppen av listan att arbeta med under iterationen. Utvecklare gör en estimering över tid som krävs och åtager sig att endast göra det som är rimligt inom iterationen. <p>Möte med team efter varje iteration där följande frågor diskuteras (Poppendieck) och enkelt dokumenteras (Björklund). Hade teamet:</p> <ol style="list-style-type: none"> 1. Nödvändig expertis? 2. Nog med information om funktionerna för att göra en rimlig tidsestimering? 3. Tillgång till resurser som behövdes? 4. Frihet, stöd och kunskap för att möta kraven? 5. Rätt uvecklingsmiljö? (Poppendieck hit) 6. Rätt underlag för att utveckla de funktioner som var av störst betydelse för kunden? (Björklund) <p>Möte med kund efter varje iteration där följande frågor diskuteras:</p> <ol style="list-style-type: none"> 1. Löser iterationen det problem som var meningen? 2. Hur kan funktionen förbättras? 	<ol style="list-style-type: none"> 1. Ja, bra här har vi snackat om A4 mallar. Hur ser en sådan mall ut? PÅ samma sätt kan man tänka sig att man har A4 rapporter i slutet av varje sprint...kort och konsist = snabbt att skriva – snabbt att läsa. Enkelt eftersom man inte har möjlighet att bli för komplext.. 2. Japp vi kör på iterationer men de kan vara dynamiska i varje projekt tex 1-4 veckor. Sätt maxlängden på en iteration tillsammans med kund. 3. Jajjemen men...kunden skall inte bara vara med vid iterationsmöten utan även vara tillgänglig på något sätt under iterationen. <p>Bra frågor – skapa en A4 mall? Checkboxar?</p> <ol style="list-style-type: none"> 5. A4 Mall..men här dokumenteras inte förändringar? De dokumenteras i backlog? Oklart? 6. Kanonbra – Detta kanske är något att ta med i review:n där frågan: är detta bra ur ett underhållsperspektiv ställs. Svaren noteras
--------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	<p>3. Hur påverkar iterationen vad ni behöver?</p> <p>4. Vad behövs för att funktionen ska kunna sättas i produktion? (Ovan Poppendieck)</p> <p>5. Genomför tester tillsammans med kund för att studera användbarhet (Björklund/Poppendieck)</p> <p>Gör teamet ansvarig för framtida underhåll av systemet</p>	och följs upp...
Synkronisering	<p>Använd dagliga builds och automatiserade tester (Poppendiecks)</p> <p>Använd en whiteboard eller annat verktyg som är synligt placerad där alla projekts status kontinuerligt uppdateras för att visa status. (Poppendiecks)</p> <p>På månadsmöte diskutera i teamet vad som är den största utvecklingsmässiga utmaningen och gör tre olika lösningar på detta samtidigt. Dokumentera. (Poppendiecks)</p>	Ja, detta kanske kan göras via visualisering – kanban? Ett whiteboard syns bara på IL. Det är en bra idé med whiteboard – kan vi använda detta för mer övergripande åtgärder ex 6 månaders möten eller kanske kaizen/måndagsmöten?
Set-baserad utveckling	<p>Presentera ett flertal prototyper för kund skapade från intial kravspec. Använd dessa för att diskutera vad kund egentligen vill ha. (Poppendiecks)</p>	Tidskrävande. Att utveckla flera förslag kan bli kostsamt för kunden. Detta är något som kunden måste vara med och fatta beslut om – kunden betalar.
Optioner / "The last responsible moment"	<p>Tillgängliggör delvis färdig designinformation för hela teamet och kunden (Poppendiecks) varje vecka. (Björklund)</p> <p>Använd utveckling i par när möjligt. (Poppendiecks)</p> <p>Gruppera tänkbara beslut som kan uppstå under iterationen i lätta och svåra beslut (Poppendiecks) på möte inför varje iteration. (Björklund) Diskutera vilken information som kan komma att behövas. (Poppendiecks)</p> <p>Identifiera efter projektavslut de mest kritiska besluten som togs. Skapa enkla regler för hur dessa beslut kan tas i</p>	<p>Bra – för att kunna stoppa prylar i ett tidigt läge – andon. Skapa en test och/eller demo server som är tillgänglig för alla intressenter... Dokument ligger på en öppen projektplats – hur får man folk att läsa detta. Kan man skapa ett dokument där några intressenter måste gå in och uppdatera ett kommentarsfält inom en viss tid...</p>

	framtiden. (Poppendiecks)	Ja, men svårt – Enklare att inför codereview – innan demo? Hur checkar man att dessa verkligen blir av? Dokument? Ta med detta i reviewen? Detta behöver vara med som en fråga på A4 mallen. Bra – detta behöver vara med som en fråga på A4 mallen.
--	---------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Djup först vs. Vidd först vid problemlösning	I möte inför projektsammansättning (Björklund) se till att människor som vill gå på djupet direkt har möjlighet att diskutera med personer som är lagda åt att tänka brett. (Poppendiecks)	Ja – det funkar....det gör vi...
----------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------

Pull system	Använd kanban (Poppendiecks) Ha dagliga 15minuters möten där följande besvaras: 1. Vad gjorde jag igår? (Poppendiecks) 2. Vad ska jag göra idag? (Poppendiecks) 3. Vad behöver jag hjälp med? (Poppendiecks) 4. Frågor som behöver diskussion hänförs till annat möte med endast involverade parter (Poppendiecks)	1. Ja 2. Nja – kan man ha detta ett par gånger i veckan?
-------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------

Köteori	På möte i slutet av varje iteration (Björklund) 1. gå igenom hur teamet spenderar sin tid. Fokusera på hur de bestämmer hur de ska spendera sin tid. Vad skulle göra detta enklare? Implementera de två bästa idéerna direkt i nästa iteration (Poppendiecks) 2. Hitta de tre längsta köerna och gör diagram över cykeltiden. Identifiera variation och trender. (Poppendiecks) 3. Sätt samman en grupp som angriper den längsta kön (Poppendiecks)	Ja – gör detta till frågor och ställ dem tillsammans med övriga frågor ovan. Cykeltid – mycket intressant – det vill jag mäta per projekt. Detta är bra för att vi skall kunna optimera flödet samt hitta tiden för uppdelning i aktiviteter. Kanske också effektivitetsgrad = nytta-tid/ waste-tid.
---------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

		Detta kan vara ett mått på att processen fungerar bättre vid en specifik åtgärd...
Förseningens kostnad	Skapa i varje projekt en nyttokalkyl som verifieras av kunden (Björklund)	Bra - egen A4 mall? Eller skall det ingå som en punkt i en annan A4 mall ex beställning?
Självbestämmande	<p>På möte efter varje iteration genomför en processkoll med teamet efter varje iteration. Fråga:</p> <ol style="list-style-type: none"> 1. Vad gör arbetet långsamt eller är i vägen för ett bra jobb? 2. Vad skulle hjälpa till att få saker att gå fortare, bättre och billigare? 3. Gör en lista med bra och dåliga practices. Bestäm vilka som kan elimineras och vilka som kan implementeras <p>På möte inför varje iteration (Björklund) se till att teamet skriver ner en eller två meningar som utgör ett tema som kan relateras till iterationens affärsvärde. Publicera denna synligt för alla och använd som beslutsstöd. (Poppendiecks)</p>	<ol style="list-style-type: none"> 1. Detta är samma A4 mall som review tidigare...bra frågor... 2. Samma fråga som vid 6 månaders möte - ex - varför gör vi detta, vad är det bra för? För vem? Mål och syfte med iteration - alla skall förstå.
Motivation	<p>Säkerställ från ledningens håll att: (Poppendiecks)</p> <ul style="list-style-type: none"> • Projekt startas med ett tydligt och intressant ändamål. (Poppendiecks) • Personer som brinner för en idé lockar andra att arbeta med den. (Poppendiecks) • ändamålet går att nå (Poppendiecks) • att teamet har tillgång till kunder (Poppendiecks) • att teamet väljer sina egna åtaganden (Poppendiecks) • Ledningen stödjer utvecklingen. Utvecklare kan komma in med saker som att de behöver en avancerad testmiljö för att nå sina mål eller behöver vissa kompetenser för att projektet ska kunna lyckas. (Poppendiecks) • skeptiker hålls borta från teamet. (Poppendiecks) • Misslyckanden tolereras (Poppendiecks) 	Skapa förutsättningar för att lyckas. En egen checklista i projektmappen på zorin med dessa frågor.

Ledarskap	Säkerställ att <i>master developers</i> kan uppstå i organisationen	Bra – vi klarar oss inte utan dessa...
-----------	---------------------------------------------------------------------	----------------------------------------

Expertis	<p>På årligt möte diskutera i organisationen vad det är för expertis som finns och hur denna kan utvecklas (Björklund)</p> <p>Låt utvecklare själva kontinuerligt utveckla kodstandarder (Poppendiecks)</p> <p>På halvårsvis möte (Björklund) be varje person i utvecklingsteamet att skriva ner ett område där teamet har låg expertis. Leta efter ett mönster i svaren och låt varje teammedlem rösta på en av dessa. Se vilken som fick flest röster och gör en plan för hur den expertisen ska göras tillgänglig för teamet. Detta kan göras genom att: (Poppendiecks)</p> <ol style="list-style-type: none"> 1. Köp en bok till alla som är intresserade och träffas en gång i veckan för att diskutera ett kapitel (Poppendiecks) 2. Hitta en guru i det som efterfrågas och låt denna para upp med olika teammedlemmar för att öka deras skicklighet. (Poppendiecks) 3. Sätt upp en kommitte på tre personer för att etablera konventioner för området i fråga. Utvärdera befintliga standarder före att konstruera egna. (Poppendiecks) 	Gött...ta upp vid 6 månaders mötet... ta med kurser och konferenser i den årliga budgeten. Säkerställ kompetens och dela kompetens -> se alltid till att fler än 1 person har kunskap om ett projekt.
----------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Upplevd integritet	<p>Chef väljer ett stort system och pratar med användare för att hitta det språk som används. Samma sak görs med utvecklarna av systemet. Efter detta får utvecklare identifiera språket i koden. Identifiera nyckelklasser som inte är representerade i ordlistan. Om det finns två eller tre olika uppsättningar med ordlistor som inte matchar så får chefen genomföra samtal med utvecklare varför det är viktigt att använda kundens språk. (Poppendieck)</p>	<p>Definiera begrepp från kund->utvecklare....</p> <p>Använd samma för att minska kommunicera...</p>
--------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------

<p>Konceptuell integritet</p>	<p>Vid utveckling av större system ha ett team möte efter var tredje iteration (Björklund) och bjud in någon av följande personer som inte brukar vara där. Det kan vara personer som: (Poppendieck)</p> <ul style="list-style-type: none"> a. Testar systemet (Poppendieck) b. Sätter systemet i produktion (Poppendieck) c. Tränar användare (Poppendieck) d. Systemansvarig (Poppendieck) e. Helpdeskpersonal (Poppendieck) f. Personer som underhåller systemet (Poppendieck) g. Personer som utvecklar eller underhåller system som använder samma data (Poppendieck) <p>Låt personerna brainstorma problem som de har om systemet som håller på att utvecklas. Skapa en grupp av människor som kan ge sig på tre av problemen som kommer upp. Följ upp efter två veckor och se om och hur problemen har lösts och upprepa sedan processen. (Poppendieck)</p>	<p>OK – men inte det första vi implementerar.</p>
<p>Refactoring</p>	<p>Sätt upp fem papper i teamrummet med enkelhet, klarhet, lämplighet, DRY och extra funktioner skrivna på dem. Be varje utvecklare att skriva ner saker som inte lever upp till standarden under korrekt rubrik. Lös problemen genom refactoring och stryk det från listan. Avsätt en dag eller två efter varje iteration för att lösa de värsta problemen. (Poppendieck)</p>	<p>Bra – proaktivitet...detta måste dock kunden godkänna eftersom det tar tid.</p>

Testning	<p>Uppskatta den genomsnittliga cykeltiden för: (Poppendieck)</p> <ol style="list-style-type: none"> Tid från att en funktion har kodats till dess att tester har körts (Poppendieck) Tid från att en funktion har kodats till dess att den är integrerad i systemet och att den automatiska testsviten har körts (Poppendieck) Tiden från att en funktion har kodats och kundtester körs (Poppendieck) Tiden från att en funktion har kodats och användbarhetstest körs (Poppendieck) Tiden från att en funktion har kodats tills att den är satt i produktion (Poppendieck) <p>Efter detta så skrivs målet för varje cykeltid ner. Gå igenom listan uppifrån och ner för att komma fram till en plan för hur varje cykel ska komma ner till målet. (Poppendieck)</p>	Bra, mät cykeltid på olika nivåer- totalt, till test, till utveckling etc... Men...hur mäter man?
Mätningar	<p>Säkerställ att defekthanteringssystem mäter information, inte prestanda (Poppendieck)</p> <p>Är defekter spårbara till utvecklaren som skapade den? Varför? Spara inte ens utvecklarens identiteter, bara defekter (Poppendieck)</p> <p>Om en utvecklarens identitet behövs så var säker på att en individuell utvecklare är den enda som ser rapporten som relaterar till hans eller hennes arbete. Aggregera alla defektrapporter. Publicera dem inte offentligt med individer listade som ansvariga. (Poppendieck)</p>	Noterat - men kanske ok att veta vem som gjort fel eftersom den personen antagligen rättar detta snabbast. Peka dock aldrig ut någon...
Kontrakt	Se till att det finns kontrakt som inte lägger hela risken hos utvecklare eller hela risken hos kunden. (Fastpris jämfört med löpande debitering) Diskutera med jurist hur kontrakt som främjar tillit kan utformas. (Poppendiecks)	Finns det mellanvägar? Förslag?

Bilaga I - Sammanfattning av Poppendieck's principer och tankeverktyg

Bilagan utgörs av en översättning och sammanfattning av Mary och Tom Poppendieck's bok "Lean software development: an agile toolkit" (2003). Jag har i denna process försökt att göra en så kortfattad men fullständig beskrivning som möjligt av principerna och tankeverktygen. För att göra det så begripligt som möjligt för läsaren har jag följt deras kapitelindelning men har valt att återge det med mina egna ord samtidigt som jag noggrant har försökt att behålla deras intentioner med beskrivningen av varje princip. Varje princip motsvarar ett kapitel i boken.

9.1.1 Princip 1 - Eliminera slöseri

9.1.1.1 Verktyg 1 - Slöseri

Tabell 5 - Slöserier i tillverkning jämfört med slöserier i systemutveckling

De sju slöserierna inom tillverkning	De sju slöserierna inom systemutveckling	Förklaring
Lager	Delvis utfört arbete	Blir snabbt inaktuellt och förhindrar annan utveckling. Binder upp resurser i investeringar som ännu inte har gett avkastning.
Extra bearbetning	Extra processer	Exempelvis pappersarbete. Behövs all dokumentation som produceras i organisationen?
Överproduktion	Extra features	Tillför komplexitet och är en potentiell felkälla. Om det inte tillför värde <i>nu</i> ska det inte göras.
Transport	Byte mellan arbetsuppgifter	Att ställa om tankeprocesser för utvecklare tar tid som kunde fokuserats på ett projekt.
Väntan	Väntan	Väntan förhindrar kunden från att uppleva värdet så snart som möjligt och ska minimeras.
Rörelse	Rörelse	Rörelse, både gällande personer och artefakter, tar tid och fokus.
Defekter	Defekter	Summan av slöseriet är slöseriets påverkan och tiden den har gått obemärkt.

Ifall något inte tillför värde till slutkunden så ska det betraktas som ett slöseri. Poppendieck's beskriver även ledarskapets aktiviteter såsom projektprioritering och releaseaktiviteter som ett möjligt slöseri eftersom de inte direkt tillför värde till produkten men i högsta grad påverkar produkten.

9.1.1.2 Verktyg 2 – Mappning av värdekedjan

Mappning sker genom att förfrågan om ett projekt kartläggs från den stund det inkommer till dess att det sätts i produktion. Det genomförs med fördel med hjälp av papper och penna genom att fråga personer i organisationen om den del av processen som de är inblandade i och genom att på en tidslinje specificera hur lång tid varje process tar.

Genom att kartlägga värdekedjan skapas möjlighet att identifiera slöseri i systemutvecklingsprocessen. Att genomföra aktiviteten skapar även insikter om hur de interna processerna fungerar eller inte fungerar.

9.1.1.3 Förslag till handling för princip 1 - Eliminera slöseri

Poppendieck föreslår att ett antal aktiviteter görs för att stödja denna process.

1. Gör en lista över de 10, 15 viktigaste aktiviteterna i organisationen. Med utgångspunkt från kunden sätt ett betyg från 1-5 på varje aktivitet. Med 1 innebär det att kunden inte bryr sig om aktiviteten och 5 att de värderar den högt. Betrakta de lägst värderade som slöseri och ta de två lägst rankade och gör en plan för hur slöseriet i dessa processer skall halveras.
2. Diskutera på nästkommande sju möten de sju olika slöserierna inom lean en i taget och diskutera huruvida dessa faktiskt är ett slöseri och om det är ett slöseri uppskatta hur lång tid det tar upp varje vecka. Efter detta bör frågan om vad som kan göras för att reducera tiden diskuteras.
3. Mappa värdekedjan i organisationen.

9.1.2 Princip 2 - Förstärk lärandet

9.1.2.1 Verktyg 3 – Återkoppling

Den traditionella vattenfallsmodellen implicerar vanligtvis att ett projekt är detaljplanerat i förväg varpå sekventiell utveckling kan ske i enlighet med traditionella projektstyrningsmodeller med klara milstolpar och en plan som är utlagd i förväg i syfte att stärka kontrollen.

Mjukvaruutveckling är dock en process som ofta kräver trial-and-error för att lösa komplexa problem. Det är därför inte tänkbart att processen som helhet skulle kunna fungera utan kontinuerlig återkoppling under tiden för genomförandet. Exempelvis handlar återkoppling om att prova idéer i kod istället för att skriva dokumentation som beskriver dessa eller att visa en kund ett förslag på lösning istället för att samla in mer krav. Denna återkoppling måste även kunna användas för att förändra processen i sig. Införs mer kontroll i processen kommer dessa återkopplingsloopar att förlängas.

9.1.2.2 Verktyg 4 – Iterationer

En övergång till *lean* inom den traditionell tillverkningsindustrin börjar oftast med att implementera ett just-in-time system där tillverkningen styrs mot kundernas order. I fallet då *lean* införs i en systemutvecklingsdomän handlar det om att börja arbeta med iterationer som styrs av snäva milstolpar. En iteration är en användbar del av det slutgiltiga systemet som designas, programmeras, testas, integreras och levereras under en kort tidsram. Genom att bygga dessa små fungerande delar av helheten möjliggörs en högre grad av återkoppling enligt beskrivning i 9.1.2.1.

Iterationer är även synkroniseringstidpunkter. Vid dessa måste varje iteration vara en del av helheten. Genom att denna synkronisering sker håller olika utvecklare kontakt med andra utvecklare och ser på det sättet lättare hur projektet som helhet utvecklas.

Varje iteration ska utgöras av en sammanhängande del av funktioner. En funktion är något som kommer från kunden i form av en användarhistoria, use case eller från en lista på krav. Den ger ett värde för kunden men som är litet nog för teamet att estimeras. Om funktionen är för stor för att göras i en iteration bryts den ner till mindre funktioner.

Varje iteration planeras genom att utvecklare estimerar funktionernas svårigheter och kunden vilka funktioner som är viktigast och till vilken kostnad. De som kunden prioriterar högst är vad som utvecklas först och hög-risk funktioner görs före andra.

Iterationer ska ske i en fixerad tidsrymd, vissa förespråkar att denna alltid ska vara lika för att skapa en rytm och andra att den ska vara situationsanpassad. Den ska dock vara lång nog för att kunna inrymma en meningsfull design-build-test cykel och kort nog för att möjliggöra regelbunden återkoppling till kunden.

Utvecklare måste vara fri att endast acceptera så mycket arbete som ryms inom den aktuella tidsrymden. Här är det viktigt att motstå kundens vilja att lägga in fler funktioner i en iteration än vad som är tidsmässigt möjligt.

Förändring av funktionalitet får endast ske inför varje iterationsstart, har iterationen påbörjats så fullföljs den. Med korta iterationer blir återkopplingstiden ändå kort. Funktionalitet ska

utvecklas genom att de funktioner med högst prioritet alltid är de som utvecklas, detta för att faktiskt utveckla vad som behövs istället för vad kunden tror att den kan komma att behöva.

En iteration är dock endast en fungerande möjlig lösning, inte automatiskt den slutgiltiga lösningen.

9.1.2.3 Verktyg 5 – Synkronisering

När flera individer arbetar med samma sak uppkommer ett behov av att synkronisera arbetet. Att bygga en del på ett visst sätt kan påverka hur en annan del kommer att fungera.

Poppendieck's rekommenderar dagliga builds och automatiska tester för att säkerställa att olika saker inte påverkar varandra. Utvecklare hämtar ut koden från ett konfigurationshanteringssystem på morgonen som de sedan jobbar emot under dagen för att i slutet av dagen checka in koden igen. När detta görs genomförs automatiska builds och tester körs. Om testerna går igenom är koden synkroniserad. De kallar detta för "the daily build and smoke test". De poängterar vikten av att testerna är automatiska och att de kan vara en del av systemet eller hela systemet beroende på omfattning. Om builds eller tester tar för lång tid att köras kommer de inte att köras, se därför till att simulera ex databasaccess eller nätverksåtkomster.

Ett annat sätt att synkronisera är att göra en mindre applikation som spänner över hela systemet vilken team kan använda som modell när de bygger sina varianter. Vet de hur en speciell sak skall gå igenom hela systemet kan resterande delar byggas på samma sätt.

Ett mer traditionellt tillvägagångssätt är att bygga en matrix som beskriver den övergripande arkitekturen och sedan låta team utveckla delar av denna. Det är då rekommenderat att börja med att designa interface för att kartlägga överlappande delar som team kan tänkas arbeta på.

9.1.2.4 Verktyg 6 – Set-baserad utveckling

Att använda set-baserad utveckling handlar om att definiera begränsningar. Detta görs genom att utveckla flera alternativ, kommunicera begränsningar och låta lösningar uppstå.

Exempelvis säg att vi frågar en kund vad den vill ha och sedan utvecklar 13 olika lösningar som används vid nästa kundmöte för att ringa in vad den egentligen vill ha varpå den av kund identifierade som riktiga lösningen utvecklas.

Detta kopplas till iterationsverktyget där de första iterationerna får användas för att skapa prototyperna.

9.1.2.5 Förslag till handling för princip 2 - Förstärk lärandet

1. Ta det svåraste problemet och hitta ett sätt att öka återkopplingen
 - I slutet av varje iteration så fråga teamet följande frågor i syfte att öka återkoppling till ledningen
 - Hade teamet rätt personalresurser?
 - Behövdes resurser som inte fanns?
 - Hur kan saker förändras för att gå fortare och bättre?
 - Vad står i vägen?
 - Öka återkopplingen till kunder genom fokusgrupper med kunden i slutet av varje iteration. Fråga följande
 - Hur bra löser denna sektion problemet?
 - Hur kan den förbättras?
 - Hur påverkar denna iteration synen på vad ni behöver?
 - Vad behövs för att sätta denna del av systemet i produktion?
 - Öka återkoppling av produkten till utvecklare på följande sätt
 - Se till att utvecklare skriver och kör tester medan de skriver kod
 - Låt analytiker, kunder eller testare skriva och köra kundtesteter medan utvecklare skriver kod. Låt utvecklare hjälpa kunderna testa om det behövs får att få dem automatiserade.
 - Se till att utvecklare observerar användbarhetstester av varje funktion allteftersom den närmar sig färdigställande för att se hur de reagerar till implementationen.
 - Öka återkopplingen inom teamet genom att
 - Göra testare till en integrerad del av utvecklingsteamet
 - Involvera operationell personal i projektets början
 - Etablera policyn att utvecklingsteamet underhåller produkten
2. Börja iterationer med en förhandlingssession mellan kunder och utvecklare. Kunder ska prioritera funktionalitet och utvecklare ska välja och åtaga sig att utveckla endast de funktioner som finns i toppen av prioritetslistan som de tror att de realistiskt kan färdigställa inom iterationens tidsrymd.
3. Sätt upp en progressionstavla i ett gemensamt utrymme så att team kan se vad som behöver göras och alla andra kan se hur projektet konvergerar.
4. Om ett system är uppdelat mellan team se till att ha en arkitektur som möjliggör att delar utvecklas så oberoende av varandra som möjligt. Hitta sätt för team att synkronisera så ofta som möjligt genom att integrera deras kod och köra automatiska tester.
5. Hitta det svåraste utvecklingsproblemet och låt teamet komma fram till tre olika lösningar. Låt teamet utforska alla tre lösningar samtidigt.

9.1.3 Princip 3 - Ta beslut så sent som möjligt

9.1.3.1 Verktyg 7 - Optioner

Agila utvecklingsprocesser kan ses som att skapa optioner som tillåter beslut att tas när kunders egentliga behov är mer känt och teknologier har fått en chans att mogna. Genom att inte ta beslut direkt utan hålla många olika vägar öppna kommer beslut att tas vid tidpunkten då det bästa underlaget finns. Det måste dock följas av insikten att optioner inte är gratis och att det krävs expertis för att veta vilka optioner som ska hållas öppna. Optioner garanterar inte framgång men möjliggör den om framtiden följer någon av optionernas riktning, de tillåter faktabaserade beslut baserade på lärande istället för spekulation.

9.1.3.2 Verktyg 8 – ”The last responsible moment”

Genom att kontinuerligt synliggöra exempelvis delvis färdig designinformation kan kortare återkopplingscykler möjliggöras. Detta är en sak som gör att beslut kan tas så sent som möjligt eftersom alla är införstådda i hur nuläget ser ut och bättre kan resonera kring beslut som behöver tas och vilken information som behövs för att ta dessa beslut. Poängen med verktyget är att beslut ska tas först när så mycket information har insamlats att det mest lämpliga beslutet kan tas.

9.1.3.3 Verktyg 9 – Djup först vs. Vidd först problemlösning

Att se på problem med bredd först innebär att betrakta problemlösning som en tratt, dvs se flera olika alternativ och se till de olika fördelar som dessa kan ge innan detaljer studeras.

Djup först involverar att ta tidiga beslut och är ett vanligt sätt för oerfarna att ta sig an problem eftersom det snabbt reducerar ett problems komplexitet. Det kan dock begränsa problemområdet för snävt för tidigt och medföra att en förändring kan göra att tiden som spenderats på att studera detaljer var bortkastad.

9.1.3.4 Förslag till handling för princip 3 - Ta beslut så sent som möjligt

1. Be människor att lista beslut som måste tas. Gruppera in dessa i två kategorier, lätta att ta och svåra att ta. Diskutera vilken slags information som skulle behövas för att omvandla ett svårt beslut till ett lätt beslut. Använd verktyg 8 för att skjuta på dessa beslut så länge som möjligt.
2. Be människor att utvärdera sin personlighet, är de benägna att använda bredd eller djup vid problemlösning? Para ihop utvecklare i par med olika benägenhet när nästa projekt planeras
3. Välj ett antal kritiska processer och utveckla enkla regler för dem så att människor kan förstå uppsåt och kan ta oberoende beslut.

9.1.4 Princip 4 - Leverera så snabbt som möjligt

9.1.4.1 Verktyg 10 – Pull system

När människor inte vet vad de ska göra så uppstår produktivetsnedgångar. Antingen kan vi tala om för människor vad de ska göra, eller så kan vi låta dem komma på det själv. När saker händer snabbt finns det inte tid att omvandla ett behov till direktiv från ledningen, genom att låta kundernas behov dra arbete istället för att med ett schema trycka ut arbete möjliggörs detta. *Lean* använder sig av *kanban* för att få till stånd detta dragningsbaserade arbete.

Startpunkten för dragningsarbete i systemutveckling är korta iterationer baserade på input från kunder i början av varje iteration. Genom att beskriva vad kunden vill ha med hjälp av användarberättelser och sätta upp dessa på ett bräde kan utvecklare själva välja vad de ska arbeta med. De tar ett kort och sätter det i kolumnen för utcheckade berättelser och bifogar sitt namn.

Det räcker dock inte att endast göra detta. Förslagsvis ska dagliga möten på 15 minuter genomföras där team-medlemmarna berättar vad de gjort igår, vad de planerar att göra idag och vad de behöver hjälp med. Saker som behöver mer detaljerad diskussion hänförs till ett annat möte där endast de involverade parterna deltar.

Om arbetet ska vara självreglerande är det viktigt att alla kan se vad som händer, även här räcker det inte med de individuella kanbankorten. Även saker som problemlistor, förbättringsidéer, systemets affärspåverkan i dagsläget för kunden, buildstatus, testbacklog m.m. är saker som behöver synliggöras.

9.1.4.2 Verktyg 11 – Köteori

Köteori handlar om att göra kötiden så kort som möjligt. Det centrala begreppet inom köteori är cykeltid, dvs snitttiden det tar för något att gå från en process början till slut. All tid som spenderas i en kö är väntetid och därför också ett slöseri.

Om tid spenderas i kö för att vänta på att ett stort stycke kod att testa kommer det också att ta lång tid att testa detta när det väl är framme. Genom att skicka mindre delar som kan testas jämnt över en tidsperiod minskas kötiden. Som exempel kan acceptanstester beskrivas. Om de integreras i iterationer kommer de att vara en naturlig del av arbetet och inte ett stort projekt i slutet som måste genomföras innan den färdiga produkten kan sättas i produktion.

9.1.4.3 Verktyg 12 – Förseningens kostnad

Fördelarna av snabbare utveckling är oftast större än vad man tror. Genom att få en bra uppskattning från marknadsföringsavdelningen om hur en försening av lansering påverkar marknadsandelar och försäljningsvolymen kan en sann bild av förseningskostnader börja kalkyleras.

För att vidare illustrera poängen om förseningens kostnader så är det inte säkert att kostanden för ett nytt verktyg kan jämföras mot hur mycket mer produktiv en utvecklare kan bli med hjälp av verktyget. Motsvarar kostnaden som förseningen skulle innebära om inte verktyget lanserades priset för verktyget är en bättre fråga att ställa för att få den verkliga kostnaden.

Utvecklare kan förstå denna kostnad genom att få en ekonomisk modell som möjliggör för dem att räkna ut vad som är affärsmässigt gångbart. Detta istället för att ledningen endast

kommunicerar kostnader och tidsramar. Om SU-organisationen inte är involverad i produktutvecklingen är det användbart att utveckla en ekonomisk modell för varje applikation utifrån kundens perspektiv. Detta möjliggör att se hur designbeslut som tas i utvecklingen kommer att påverka kundens affärer. Om en utvecklare ska välja på att spara en veckas arbetstid, 100000kr eller att lägga till nya funktioner är det svårt att förstå hur detta påverkar kunden. Kan samma beslut tas utifrån ett enhetligt mått, som kronor, blir det enklare att välja.

9.1.4.4 Förslag till handling för princip 4 - Leverera så snabbt som möjligt

1. Skapa ett enskilt ställe där alla intressenter kan se:
 - a. Målet för nuvarande iteration
 - i. Vad som har gjorts
 - ii. Vad som görs
 - iii. Vad som ska göras
 - b. Projektets mål
 - i. Vad som gjorts för att nå detta
 - ii. Vad som behöver göras för att nå detta
2. I slutet på iterationen gå igenom processen och förstå hur alla vet vad de ska göra. Be teamet att fokusera på sättet de bestämmer hur de spenderar sin tid. Vad skulle göra möjliggöra för dem att ta snabbare och bättre beslut om vad som är viktigt? Implementera de två bästa idéerna i nästa iteration
3. Hitta de tre längsta köerna och gör ett diagram över cykeltiden för varje jobb allteftersom det går genom varje kö. Är variationen hög eller låg? Är det en stigande eller fallande trend?
4. Välj ut kön som representerar den största flaskhalsen och sätt samman en grupp som fokuserar på att minska denna kö.
5. Be ekonomiavdelningen att avsätta en redovisningsresurs till varje utvecklingsteam och låt denna jobba med teamet för att sätta samman en ekonomisk modell som visar kostnaden för försening, kostnaden för minskade funktioner, förvaltning osv.

9.1.5 Princip 5 - Bemyndiga teamet

9.1.5.1 Verktyg 13 – Självbestämande

För att *lean software development* ska kunna fungera är det av yttersta vikt att utvecklare är de som sätts i fokus och att de får rättigheten att kontinuerligt utveckla hur de arbetar. Medarbetare måste kunna tala om för ledningen hur den kan låta dem göra sitt jobb, istället för att ledningen talar om för medarbetare hur de ska göra. Genom att låta människor bli ansvariga för sitt eget arbete och genom att den som kommer fram till förslag till förbättring blir ansvarig för att verkställa detta skapas detta självbestämmande. En viktig funktion för ledarskapet blir att direkt säga ja eller nej till förbättringsförslag.

9.1.5.2 Verktyg 14 – Motivation

Människor behöver känna att det de gör har ett värdefullt ändamål och att de inte endast betar av saker på en att-göra-lista. Det finns ett antal saker som man kan göra för att nå detta. Exempelvis så är det bra att börja med ett tydligt och intressant ändamål och låta personer som brinner för en idé locka andra att arbeta med den eftersom de då kommer att känna ett större värde.

Utvecklare måste ha tillgång till kunder att diskutera med och de måste få välja sina egna åtaganden utifrån diskussioner med kunderna. Ledningens roll blir till stor del att stödja utvecklingen. Utvecklare ska kunna komma in med saker som de behöver för att nå sina mål, vilket kan vara exempelvis en viss kompetens eller ett visst verktyg.

Motivation bygger på ett antal faktorer. Tillhörighet till gruppen är en av dessa och innebär att alla är förtrodda med målet är och att gruppen som helhet är vad som gör något till en framgång eller ett misslyckande, att skapa grupper av individer istället för gruppmedlemmar är ett säkert sätt att ta död på motivation.

Det är viktigt att ledarskapet har så stort förtroende att även misslyckanden kan tolereras, för att åstadkomma nya saker kommer ibland misstag att ske men i det långa loppet kommer misstagen att leda till nya framgångar.

För att lyckas måste människor känna att de har kompetens att utföra sitt jobb. Det blir ledarskapets ansvar att se till att det som delegerats faktiskt fungerar och ge stöd som möjliggör detta.

Varje iteration gör att teamet känner att de har gjort framsteg. Framsteg är värda att fira, små framsteg firas genom att gratulera varandra, större genom att göra något roligt tillsammans för en stund. Betydelsefulla framsteg firas publikt och så snart som möjligt, gärna direkt. Projekt måste därför ha tydliga mätinstrument som gör att framsteg kan synliggöras för alla.

9.1.5.3 Verktyg 15 – Ledarskap

Bakom varje passionerat team finns en passionerad ledare. Ledarskap i *lean software development* handlar om att inse att det är bättre att använda en stor grupps talanger istället för att försöka kontrollera arbetet de utför. Ledare ska vara passionerade och motivera andra, parallellt dras ofta till sportvärlden där en coachs uppgift är att låta sitt lag nå sin maximala potential. Vi pratar om *master developers* som med hjälp av sin tekniska expertis inom domänen, kommunikationsförmåga och kännedom om kunden blir en informell ledare som resterande utvecklare respekterar och lyssnar på. Dessa *master developers* kan men behöver

inte utses i början av ett projekt, ofta framträder de av sig själv en bit in i projektet. Var vaksam med att utse dessa ledare, de är inte effektiva om de inte respekteras av teamet.

Ledare kommer endast att blomstra i en organisation som vill ha dem där, för att få bra ledare måste organisationen värdesätta dessa högre än dokumentation, planering och konformism till plan.

9.1.5.4 Verktyg 16 – Expertis

Om ett företag vill behålla en ”competitive advantage” måste de utveckla expertis inom ett område som inte finns bland andra företag. Normala matrisorganisationer utvecklar expertis inom klart definierade områden som är ansvariga för sina egna delar. Att få denna organisationsstruktur att fungera handlar om att sättet som ledarskapet ser på sitt jobb. Framgångsrika organisationer har ledare som ser sitt jobb som mentorer och lärare. De ser till att det finns *master developers* som hjälper till att lära upp juniora medarbetare med rätt stöd och coachning. De ser sig som motivatorer och möjliggörare som låter sina medarbetare bidra med expertis till olika utvecklingsprojekt.

Systemutveckling behöver standarder. Kodstandarder, inchecknings och utcheckningsstandarder m.m. Den samlade expertisen inom en grupp är oftast vad som utvecklar dessa standards, om vi vill veta vart expertis saknas så kan vi se vart det saknas standards.

9.1.5.5 Förslag på handling för princip 5 - Bemyndiga teamet

1. Genomför en processkoll med teamet efter varje iteration. Fråga:
 - a. Vad gör arbetet långsamt eller är i vägen för ett bra jobb?
 - b. Vad skulle hjälpa till att få saker att gå fortare, bättre och billigare?
2. Gör en lista med bra och dåliga practices. Bestäm vilka som kan elimineras och vilka som kan implementeras. Genomför detta efter varje iteration.
3. Säkerställ att utvecklarteamet skriver ner målet med iterationen inför varje iterations början. Det ska vara en eller två meningar som ger ett tema som kan relateras till affärsvärdet den ska leverera. Sätt upp målet synligt och referera till det när teamet brottas med svåra beslut.
4. Använd pair programming eller design reviews. Fokusera på lärande och delande av kunskap.
5. Be varje person i utvecklingsteamet att skriva ner ett område där teamet har låg expertis. Leta efter ett mönster i svaren och låt varje teammedlem rösta på en av dessa. Se vilken som fick flest röster och gör en plan för hur den expertisen ska göras tillgänglig för teamet. Detta kan göras genom att:
 - a. Köp en bok till alla som är intresserade och träffas en gång i veckan för att diskutera ett kapitel
 - b. Hitta en guru i det som efterfrågas och låt denna para upp med olika teammedlemmar för att öka deras skicklighet.
 - c. Sätt upp en kommitté på tre personer för att etablera konventioner för området i fråga. Utvärdera befintliga standarder före att konstruera egna.

9.1.6 Princip 6 - Bygg in integritet

Produktintegritet har två dimensioner. Upplevd integritet och konceptuell integritet.

9.1.6.1 Verktyg 17 - Upplevd integritet

Upplevd integritet är summan av balansen mellan funktion, användbarhet, pålitlighet och ekonomi. Ett exempel är program eller webbsidor som känns som att de var gjorda för dig och fungerar som du tycker att de ska göra.

För att nå upplevd integritet är det viktigt att alltid komma ihåg att det är kunden som avgör om produkten faktiskt har upplevd integritet eller inte. När krav samlas in från kunden är de inte slutgiltiga, när mer krav samlas in kommer kunden att börja förstå vad den egentligen vill ha och detta förändrar vad den kommer att se för upplevd integritet. Att säkra att kunden upplever integritet är därför väldigt viktigt och kan säkras med hjälp av ett antal olika verktyg.

Mindre system ska utvecklas av ett enskilt team som har direktkontakt med den som ska bedöma integriteten. Teamet ska använda korta iterationer och visa resultatet av varje iteration för ett brett spektra av människor som kan känna igen integritet när de ser den. Detta görs för att säkra återkoppling. Att använda kundtester ger bra kommunikation mellan kunden och utvecklaren. (Se Verktyg 20 – Testning för en beskrivning av kundtester)

Genom att använda modeller som både utvecklare och kunden förstår och som är baserade på kundens språk får vi ett bra underlag för kommunikation mellan utvecklare och kunden.

Det finns ett antal olika modeller som har visat sig vara bra för att stödja detta.

- Konceptuell domänmodell - Kan vara en klassmodell över de grundläggande entiteterna i systemet. Det kan vara händelser, dokument, transaktioner m.m. Den måste inkludera både nyckelkoncepten i användarnas uppfattning och relationerna mellan dessa koncept. Fokusera på idéer och orosmoment istället för detaljer och uttömlighet, modellen ska fånga kundens uppfattning av systemdomänen.
- Ordlista som definierar termerna i domänmodellen på kundens språk.
- Use case modell - En dynamisk syn på systemet som är användbar för att hitta kunskap om vad användbarhet betyder i domänen. Organiserar och detaljerar kundens mål och delmål för att interagera med domänmodellen och driver arbetsflödet och riktningen.
- Vilka multipler och kvantifierare finns det i systemet? Utvecklare måste vara medvetna om hur många användare systemet ska kunna hantera, acceptabla responstider, tillgänglighet, vinstprognostiseringar m.m.

9.1.6.2 Verktyg 18 – Konceptuell integritet

Den konceptuella integriteten är hur väl ett systems centrala koncept fungerar som en sammanhängande helhet, den uppnår ett stadie av balans mellan flexibilitet, underhållbarhet, effektivitet och respons. Som exempel kan vi ta ett system som har två olika moduler för att skapa en faktura beroende på om det är en faktura för internleverans eller extern leverans. Upplevelsen blir att det känns som att de har utvecklats var för sig utan vetskap om varandra. Konceptuell integritet behövs för att kunna ha en upplevd integritet.

För att sammanfoga alla de delar som ett projekt består av krävs det kommunikation mellan de grupper som jobbar med de olika delarna. Det behövs en integrerad problemlösning som sammanfogar delarna till en helhet. Med integrerad problemlösning menas att vi förstår att problemet och dess lösning händer samtidigt, inte sekventiellt. Vidare innebär det att information släpps på ett tidigt stadie i små paket, inte sent i ett stort paket. Det innebär även att information flyter i två riktningar, inte bara en och att det föredragna sättet att kommunicera är verbalt istället för med dokument.

Utan denna sorts problemlösning så bestämmer designers enskilt vilken kombination av funktioner och kapacitet som bäst kommer leverera värde till kunden. Skickar vi den kompletta designen till en kund kommer den ha svårt att ta ställning till de enskilda delarna som faktiskt utgör helheten.

9.1.6.3 Verktyg 19 – Refactoring

Ett system är sällan perfekt efter första gången. Designbeslut som passade bra i en fas av systemets framväxt kanske passar sämre in i helheten. Bra design utvecklas över en längre tidsperiod, inte i planeringsfasen av ett projekt. Ett systems interna struktur är något som vi behöver utveckla kontinuerligt för att den inte ska bli mismatchad med de nyare delarna. Det är en god idé att bygga in system så att de har olika moduler som kan användas på olika ställen. Ex input av adresser som kan ta olika värden är något som kan behövas på flera ställen i ett system om lätt kan bli mismatchade om de kodas var för sig varje gång. Om detta skulle hända så mister systemet konceptuell integritet.

När ett system börjar mista sin konceptuella integritet är det dags att genomföra refactoring. Det finns ett antal tecken på att den konceptuella integriteten börjar minska vilken utvecklare bör vara vaksamma mot:

- Koden mister enkelhet och blir komplex
- Koden mister klarhet, det ska gå att förstå vad en bit kod gör även utan kommentarer
- Lämpligheten för att använda lösningen minskar. Ex GUI börjar bli svårt att använda, prestandan börjar nå en oacceptabel nivå
- Avvikelser från DRY. Kod ska endast finnas på ett ställe
- Det finns extra funktioner i systemet som aldrig används vilket är ett slöseri eftersom de behöver underhållas

Refactoring ses på samma sätt som att stoppa linjen i traditionell tillverkning, när det finns problem i koden som stör flödet i det normala arbetet är det dags att ta tag i problemet direkt. Teamet måste då gå till roten och lösa problemet innan det går att fortsätta jobba som vanligt.

9.1.6.4 Verktyg 20 – Testning

När utvecklare skriver kod ska testning följa. Syftet med att testa är att se till att den individuella delen fungerar som den ska, följer den övergripande designen och att den passar in i helheten.

Normalt talar vi om acceptanstestning för att säkra att systemet fungerar på det sätt som kunden vill. Dessa ligger dock oftast i slutet av utvecklingsprocessen och Poppendieck's vill istället prata om kundtester vilka utgör en integrerad del av hela processen. Genom att kontinuerligt involvera kundtester får vi direkt återkoppling och en större möjlighet att använda oss effektivare av set-baserad design, fördröjda beslut och refactoring. Kundtester kan även vara ett potentiellt alternativ till dokumenterade kravspecifikationer. Genom att inte anse att ett krav är komplett förens det har omvandlats till ett kundtest säkras det att kravet är

genomtänkt och faktiskt motsvarar något som kunden vill ha. På samma sätt kan vi säkra designen genom att utveckla tester för den samtidigt som utvecklare kommer överens om hur den ska fungera.

Utveckling är en iterativ process som består av flera experiment. Utvecklare kommer hela tiden att hitta på sätt att testa så att det som de kodar faktiskt fungerar, eftersom processen ändå görs är det bra att se till att det finns tester skrivna för varje mekanism som en utvecklare kodar. Tester ska automatiseras så mycket som det går och vara en del av den dagliga builden. Om det fattas tid för att arbeta med dessa tester så ska den tas från tid som spenderas med kravdokumentation.

9.1.6.5 Förslag till handling för princip 6 - Bygg in integritet

1. Välj ett nuvarande system och se om det har ett allmänt språk. Prata med användare och skriv ner en ordlista utifrån samtalet om systemet, se vilka ord som används. Prata efter det med utvecklarna och se om de använder samma ord för att prata om systemet. Be efter det utvecklarna att identifiera orden i koden och slutgiltigen se om det finns några nyckelklasser i systemet som inte är representerade i ordlistan. Om det finns två eller tre olika uppsättningar med ord så behövs det en diskussion med utvecklarna om varför det är viktigt att använda kunders språk i utvecklingen och i samtal med varandra.
2. Ha ett team möte och bjud in någon av följande personer som inte brukar vara där. Det kan vara personer som:
 - a. Testar systemet
 - b. Sätter systemet i produktion
 - c. Tränar användare
 - d. Systemansvarig
 - e. Helpdeskpersonal
 - f. Personer som underhåller systemet
 - g. Personer som utvecklar eller underhåller system som använder samma dataLåt personerna brainstorma problem som de har om systemet som håller på att utvecklas. Skapa en grupp av människor som kan ge sig på tre av problemen som kommer upp. Följ upp efter två veckor och se om och hur problemen har lösts och upprepa sedan processen.
3. Sätt upp fem papper i teamrummet med enkelhet, klarhet, lämplighet, DRY och extra funktioner skrivna på dem. Be varje utvecklare att skriva ner saker som inte lever upp till standarden under korrekt rubrik. Lös problemen genom refactoring och stryk det från listan. Avsätt en dag eller två efter varje iteration för att lösa de värsta problemen.
4. Uppskatta den genomsnittliga cykeltiden för:
 - a. Tid från att en funktion har kodats till dess att tester har körts
 - b. Tid från att en funktion har kodats till dess att den är integrerad i systemet och att den automatiska testsviten har körts
 - c. Tiden från att en funktion har kodats och kundtester körs
 - d. Tiden från att en funktion har kodats och användbarhetstest körs
 - e. Tiden från att en funktion har kodats tills att den är satt i produktionEfter detta så skrivs målet för varje cykeltid ner. Gå igenom listan uppifrån och ner för att komma fram till en plan för hur varje cykel ska komma ner till målet.

9.1.7 Princip 7 - Se helheten

9.1.7.1 Verktyg 21 – Mätningar

Inom *lean* brukar frågan varför ställas fem gånger när ett problem uppstår för att komma fram till den egentliga orsaken. Om ett fel uppstår räcker det inte med slutsatsen att testaren var stressad när det egentligen kan handla om att teamledaren är rädd att inte nå deadline och därför har satt extra tigha deadlines för testarna.

Vi tenderar att vilja bryta ner stora komplexa saker i mindre hanterbara bitar. Men när vi bryter ner allting och optimerar varje subprocess för sig är det inte säkert att vi kommer få ett optimalt resultat i helheten. Det spelar ex ingen roll om vi gör extremt snabba sökningar när vi inte har en databas som är snabb nog för att kunna leverera data som behövs. Eller om vi vill optimera användandet av testare för att bättre utnyttja företagets resurser så verkar det logiskt att se till att de har 100 % beläggning. Problemet är då att det samlas jobb i en flaskhals hos testarna. Någon väntar på att testen ska köras, relatera detta tillbaks till vad väntan kostar.

För att undvika detta vill vi gärna hitta ett sätt att mäta helheten. Ofta görs detta genom att standardisera arbete och mäta hur väl människor följer det som har bestämts. Eller så planerar vi i detalj för att mäta variationer och konformitet till planen. Eller så bryter vi ner helheten till delar och försöker mäta dem var för sig.

För att mäta helheten behöver vi gå upp ett steg, vi behöver slå ihop saker istället för att bryta isär dem. Ex mät en grupp istället för en individ, gå ifrån att mäta prestanda till att mäta information. Mät defekter på grupp eller systemnivå och uppmuntra hela team att lösa problemen istället för den individ som till synes är ansvarig för defekten. Det behöver inte vara en individs fel att en defekt uppstår när de bakomliggande orsakerna troligtvis är vad som gjort att defekten uppstått.

9.1.7.2 Verktyg 22 – Kontrakt

Det är viktigt för företag att de avtal man ingår med andra parter hålls. Normalt brukar det säkras genom vattentäta kontrakt där varje detalj är avtalad i detalj för att se till att ingen av parterna utnyttjar den andra. Toyota lyckades med något annorlunda, de lyckades utveckla en tillit. Detta medförde exempelvis att de fick tillgång till hemligstämplat material eftersom de inte trodde att det skulle läcka ut till någon konkurrent. Underleverantörer visste att Toyota gjorde fler affärer med samma tillverkare 90 % av gångerna. Toyota lånade även ut lean experter till underleverantörer utan att använda den information som de fick för att sätta press på dem.

Avtal som specificerar en fast summa för ett system inkluderar oftast inte enligt Poppendieck's de kostnader som uppstår när den färdiga produkten behöver förändras. I den traditionella biltillverkningen så var kostnaden för gjutformar ungefär hälften av den totalt kostnaden för en ny bilmodell. Eftersom designen traditionellt låstes så kunde inte förändringar göras före det att formen var klar och de kunde se hur resultaten blev. Det kostade då väldigt mycket pengar att göra förändringar av dessa och rigorösa förändringsprocesser skapades för att minska detta. Vad som egentligen händer är att det som inte fanns med i avtalet från början nu är ett tillfälle för tillverkaren att få in de pengar som förlorats i det fasta priset. Genom att ta beslut så sent som möjligt och samverka med leverantörer kunde Toyota göra förändringar under tiden formarna producerades vilket ledde

till en bättre slutprodukt. Detta illustrerar hur kontrakt som är ämnade att skydda de enskilda parterna och kontrakt som baseras på samverkan.

Kontrakt som grundar sig på fasta priser lämnar över en stor del av risken på tillverkaren. Produkten ska levereras enligt kontraktet, frågan är om det kunden egentligen vill ha är vad som hamnar i kontraktet. Erfarna tillverkare kommer att justera in risken i priset de vill ha för en produkt under en fast kostnad, oerfarna kommer inte att göra det och således lämna ett lägre kostnadsförslag. I slutändan kommer det för tillverkaren att handla om att skydda sina egna intressen på bekostnad av kunden. Det är svårt att få tillit att växa under sådana omständigheter.

Som en motsats till detta har vi tids och materialsbundna kontrakt där risken hamnar hos kunden. Kunden betalar för den tid och det material som tillverkaren använder under projektet. Det är tänkbart att det kommer ta längre tid för en tillverkare att åstadkomma ett resultat under denna kontraktsform då de tjänar mer pengar ju längre tid projektet tar. Det uppkommer även extra kostnader i form av kontroll för att se till att tillverkaren faktiskt utför det arbete som den har åtagit sig under kontrollerade och effektiva former. Den iterativa utveckling som har presenterats hitintills ger en kontroll åt kunden då den i slutet av varje iteration har något fast att se på och prova. När systemet växer fram iterativt så finns det en kontroll inbyggd då planering kan göras utifrån dessa faser.

Kontrakt med flera stadier försöker hantera riskerna som finns i kontrakt med fasta priser genom att matcha riskerna till pengar spenderade över tiden. Det finns två typer av sådana kontrakt, de som ska leda till ett stort kontrakt med ett fast pris och de som behåller de flera stadierna genom hela livslängden.

De som mynnar ut i ett stort fastpriskontrakt involverar oftast två korta kontrakt för att bekanta sig med problemet i syfte att möjliggöra en välgrundad kostnads kalkyl för resterande del av projektet. Detta involverar oftast endast en tillverkare då det skulle vara krångligt med flera som gjorde samma sak i en organisation. Eftersom problemet när de läggs ett förslag på lösning fortfarande är väldigt statistiskt finns det små möjligheter att avvika från kontraktet när det väl är fastlagt.

Den andra typen av kontrakt som behåller sina faser genomgående presenterar ett bra tillfälle för agil utveckling. De är dock inte utan risker och den största risken är att både tillverkare och kund närsomhelst kan avbryta projektet. Båda parter blir beroende av varandra och detta kräver tillit. Detta kan hanteras genom att leverera värde i varje fas av kontraktet. Implementera det som ger störst värde först så minskas risken av ett tidigt avslut av projektet.

9.1.7.3 Förslag till handling för princip 7 - Se helheten

1. Säkerställ att defekthanteringssystem mäter information, inte prestanda
 - a. Är defekter spårbara till utvecklaren som skapade den? Varför? Spara inte ens utvecklarens identiteter, bara defekter
 - b. Om en utvecklarens identitet behövs så var säker på att en individuell utvecklare är den enda som ser rapporten som relaterar till hans eller hennes arbete. Aggregera alla defektrapporter. Publicera dem inte offentligt med individer listade som ansvariga.

Kom på hur scope görs optional. Be jurister att gå igenom litteraturen om metoder för att hitta avtalsformer som ger lämpligt skydd för företaget och kunden utan att använda en fast-scope spec.

Högskolan i Borås är en modern högskola mitt i city. Vi bedriver utbildningar inom ekonomi och informatik, biblioteks- och informationsvetenskap, mode och textil, beteendevetenskap och lärarutbildning, teknik samt vårdvetenskap.

På **institutionen för data- och affärsvetenskap (IDA)** har vi tagit fasta på studenternas framtida behov. Därför har vi skapat utbildningar där anställningsbarhet är ett nyckelord. Ämnesintegration, helhet och sammanhang är andra viktiga begrepp. På institutionen råder en närhet, såväl mellan studenter och lärare som mellan företag och utbildning.

Våra **ekonomiutbildningar** ger studenterna möjlighet att lära sig mer om olika företag och förvaltningar och hur styrning och organisering av dessa verksamheter sker. De får även lära sig om samhällsutveckling och om organisationers anpassning till omvärlden. De får möjlighet att förbättra sin förmåga att analysera, utveckla och styra verksamheter, oavsett om de vill ägna sig åt revision, administration eller marknadsföring. Bland våra **IT-utbildningar** finns alltid något för dem som vill designa framtidens IT-baserade kommunikationslösningar, som vill analysera behov av och krav på organisationers information för att designa deras innehållsstrukturer, bedriva integrerad IT- och affärsutveckling, utveckla sin förmåga att analysera och designa verksamheter eller inrikta sig mot programmering och utveckling för god IT-användning i företag och organisationer.

Forskningsverksamheten vid institutionen är såväl professions- som design- och utvecklingsinriktad. Den övergripande forskningsprofilen för institutionen är handels- och tjänsteutveckling i vilken kunskaper och kompetenser inom såväl informatik som företagsekonomi utgör viktiga grundstenar. Forskningen är välrenommerad och fokuserar på inriktningarna affärsdesign och Co-design. Forskningen är också professionsorienterad, vilket bland annat tar sig uttryck i att forskningen i många fall bedrivs på aktionsforskningsbaserade grunder med företag och offentliga organisationer på lokal, nationell och internationell arena. Forskningens design och professionsinriktning manifesteras också i InnovationLab, som är institutionens och Högskolans enhet för forskningsstödande systemutveckling.



HÖGSKOLAN I BORÅS

VETENSKAP FÖR PROFESSION

BESÖKSADRESS: JÄRNVÄGSGATAN 5 · POSTADRESS: ALLÉGATAN 1, 501 90 BORÅS
TFN: 033-435 40 00 · E-POST: INST.IDA@HB.SE · WEBB: WWW.HB.SE/IDA