CompLex

Senter for rettsinformatikk / Avdeling for forvaltningsinformatikk

Dan Sørensen

Kontraktsregulering av smidig programvareutvikling

Hvilke kontraktsmekanismer kan bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes?

3/2014

CompLex 3/2014

Dan Sørensen

Kontraktsregulering av smidig programvareutvikling

Hvilke kontraktsmekanismer kan bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes?

Senter for rettsinformatikk
Avdeling for forvaltningsinformatikk
Postboks 6706 St Olavs plass
0130 Oslo

Henvendelser om denne bok kan gjøres til:
Senter for rettsinformatikk
Postboks 6706 St. Olavs plass
0130 Oslo
Tlf. 22 85 01 01
www. jus.uio.no/iri/

ISBN 978-82-72261-56-5
ISSN 0806-1912

Grafisk produksjon: 07 Media AS – 07.no

Innhold

1 Problemstilling og metode

1.1 Innledning

I 2011 benyttet 81 prosent av it-bransjen i Norge såkalt smidig programvareutvikling.[1] Internasjonalt er tilsvarende tall 88 prosent i 2013.[2] Forskning og bransjeerfaring har vist at smidig programvareutvikling blant annet reduserer risikoen for kostnadsoverskridelser,[3] gir programvaren høyere kvalitet og øker forutsigbarheten i prosjektene.[4] It-bransjen har i all sin tid har vært plaget med store tids- og kostnadsoverskridelser, både nasjonalt og internasjonalt.[5] Da er det ikke overraskende at smidig programvareutvikling har spredt seg raskt i bransjen i løpet av de siste 10 årene. Likevel står utfordringene fortsatt i kø for it-bransjens utviklingsprosjekter.

For å lykkes med smidig programvareutvikling er det flere forutsetninger som må være på plass. Disse forutsetningene har de siste årene blitt drøftet inngående, både fra teknologisk, organisatorisk og ledelsesperspektiv. Omfanget av litteratur og stoff på disse fagområdene bekrefter dette. Det kontraktsrettslige perspektiv er derimot lite belyst. Kontrakten er et viktig styringsverktøy og skal være konfliktforebyggende ved gjennomføringen av prosjekter hvor noe skal tilvirkes.[6] Hvordan kontrakten utformes blir derfor en viktig forutsetning for å lykkes med smidig programvareutvikling.

Smidig programvareutvikling er en utviklingsmetode som skiller seg radikalt fra de tradisjonelle plandrevne utviklingsmetodene, eller de såkalte sekvensielle utviklingsmetodene. Det tradisjonelle utgangspunktet er at alle fasene i prosjektgjennomføringen skjer sekvensielt, en ny fase starter ikke før forrige fase er avsluttet. I en smidig utviklingsmetode utarbeides for eksempel ikke en ferdig kravspesifikasjon før programvareutviklingen starter. Programvaren utvikles i mindre inkrementer (små delleveranser) med typisk varighet på to til seks uker. Hvert inkrement har gjennomført en kravspesifisering, programvareutvikling, testing og aksept.

De tradisjonelle tilvirkningskontraktene i både entreprise, fabrikasjon og programvareutvikling er utformet med utgangspunkt i en *plandreven metode*. Veldig forenklet er forløpet i disse kontraktene som følger: Spesifikasjonen av

1 Brevik (2013) s. 20
2 VersionOne (2014a) s. 2
3 Se Moløkken-Østvold (2005) s. 765 og Ambler (2012) s. xx
4 Larman (2004) s. 52-53
5 Se for eksempel Moløkken-Østvold (2004)
6 Torvund (1997) s. 42

produktet blir gjort ferdig før leverandøren starter tilvirkningen og prosjektet avsluttes med kundens aksept. Skal spesifikasjonen endres underveis i skjer det ved hjelp av formaliserte endringsprosedyrer. Slike kontrakter skal blant annet sikre at leverandøren oppfyller alle kravene som er spesifisert ved kontraktsinngåelsen.

I en kontrakt for smidig utviklingsmetode vil ikke en slik tankegang være aktuell, siden det ikke finnes en kravspesifikasjon ved kontraktsinngåelsen. Dermed forutsettes det at kunden har en aktiv rolle ved gjennomføringen og tar beslutninger underveis. Dette endrer den risikofordelingen mellom partene som de tradisjonelle tilvirkningskontraktene legger til grunn. Hvis det ikke benyttes kontrakter som er tilpasset en smidig utviklingsmetode øker prosjektrisikoen.[7] Det blir også hevdet at det å bruke de tradisjonelle kontraktene ved smidige utviklingsprosjekter er en viktig årsak til tids- og kostnadsoverskridelser.[8]

Selv om smidige utviklingsmetoder har blitt brukt i vel 15 år har det ikke vært utviklet kontrakter som er tilpasset denne måten å jobbe på. Det er først de siste to-tre årene tatt initiativ som har resultert i standardiserte kontrakter som skal avhjelpe denne situasjonen. Direktoratet for forvaltning og IKT (Difi) har i flere år arbeidet med en ny statens standardavtale for smidige prosjekter (SSA-S), og en slik avtale ble lansert i januar 2014.[9] Den Norske Dataforening (DND), Digitaliseringsstyrelsen i Danmark og IT & Telekomföretagen i Sverige har også lansert smidige kontrakter de siste par årene.[10]

It-bransjen i Norge hadde i 2010 en årlig omsetning på 202 milliarder kroner.[11] Derfor er det av stor økonomisk betydning at programvareutviklingsprosjektene gjennomføres uten havari. Et av formålene med denne oppgaven er å kunne bidra i arbeidet med å utforme kontrakter som kan sikre vellykket gjennomføring av smidig programvareutvikling.

1.2 Problemstilling

Problemstillingen jeg skal drøfte i denne oppgaven tar utgangspunkt i den fundamentale forskjellen mellom en smidig utviklingsmetode og en plandreven utviklingsmetode. Denne forskjellen gjør at tradisjonelle kontraktsmekanismer, som er tilpasset plandrevne utviklingsmetoder, er lite egnet for regulering av smidig programvareutvikling. Som et bakteppe for problemstillingen ligger det såkalte *Manifestet for smidig programvareutvikling*.[12] Dette manifestet, med de

7 Larman (2010) s. 512
8 Atkinson (2013)
9 Difi (2014)
10 Digitaliseringsstyrelsen (2012); IT&Telekomföretagen (2012); Den Norske Dataforening (2013a)
11 Se kapittel 1.4 i Meld. St. 23 (2012-2013)
12 Se Beck (2001a)

tilhørende tolv prinsippene,[13] ligger som et fundament for alle smidige utviklingsmetoder. Manifestet er presentert i tabell 1.1.

Tabell 1.1: Manifestet for smidig programvareutvikling

Manifestet for smidig programvareutvikling
Vi finner bedre måter å utvikle programvare på *ved å gjøre det selv og ved å hjelpe andre med det.* *Gjennom dette arbeidet har vi lært oss å verdsette følgende:*
Personer og samspill *fremfor prosesser og verktøy* ***Programvare som virker*** *fremfor omfattende dokumentasjon* ***Samarbeid med kunden*** *fremfor kontraktsforhandlinger* ***Å reagere på endringer*** *fremfor å følge en plan*
Dette vil si: Selv om punktene som står til høyre har verdi, *så verdsetter vi punktene til venstre enda høyere.*

Manifestet ble utformet og undertegnet i 2001 av 17 eksperter med lang erfaring fra programvareutvikling og prosjektledelse. Målsettingen med manifestet var å tydeliggjøre hvilke grunnleggende verdier og tankesett som ligger bak smidige utviklingsmetoder, samt å ha en felles plattform.[14] Punktene gir et godt bilde av forskjellen mellom en plandreven utviklingsmetode (det som står til høyre), og en smidig utviklingsmetode (det som står til venstre). Likevel er det viktig å understreke at det som står høyre har verdi, men at det som står til venstre verdsettes høyere. En kontrakt for smidig programvareutvikling må derfor utformes med utgangspunkt i dette manifestet. Jeg kommer nærmere tilbake til smidig programvareutvikling, men som et bakteppe for problemstillingen er det nødvendig med en kort forklaring av manifestets innhold.

Personer og samspill handler om teamarbeid ved utviklingen av programvare. De ulike enkeltpersonene skal fungere sammen på best mulig måte for å være effektive og skape et produkt med høy kvalitet. I tillegg skal personene som er involvert ta større ansvar for hvordan forretningsmålene skal realiseres, og finne løsninger som alle interessenter i prosjektet verdsetter. Prosesser og verktøy kan ikke erstatte et godt fungerende utviklingsteam. Prosjektets interessen-

13 Se Beck (2001b)

14 Jf. Larman (2003) s. 54

ter må ha *tillit* til at teamet selv organiserer seg slik at de optimale løsningene velges på en mest mulig effektiv måte

Programvare som virker er vel noe alle utviklingsprosjekt har som mål, uansett hvilken metode som brukes. I denne sammenhengen representerer dette de hyppige leveransene av programvare, som kunden og brukerne kan evaluere. Løsningen «gror» og utvides og forbedres for hver leveranse, og eventuelle problemer og utforutsette ting *oppdages tidlig*. Motsetningen er en lang prosess med dokumentasjon og spesifikasjon av alle krav og behov før programmeringen tar til. De hyppige leveransene legger til rette for en løpende empirisk kontroll av fremdriften.

Samarbeid med kunden fremfor kontraktsforhandlinger er et punkt som ofte kan misforstås, og tolkes som at kontrakter ikke er viktige. Kontrakter er viktige også ved smidig programvareutvikling, men et tett samarbeid mellom partene vil gi prosjektet bedre muligheter for å nå ønskede resultater. I komplekse prosjekter er det kanskje ikke komplekse og detaljregulerte kontrakter som er ønskelig, det er viktigere å legge til rette for et godt samspill mellom kunde og leverandør. Dette gir også muligheter for å redusere kontraktens omfang, og dermed gjøre kontrakten mer forståelig og tilgjengelig for alle interessenter i prosjektet. På den måten kan kontrakten også fungere bedre som styringsverktøy og bidra til at alle har den samme forståelsen av hvilke mål prosjektet har, og hva som skal til for å nå målene.

Å reagere på endringer er noe smidig utviklingsmetoder spesielt er laget for. I en plandreven utviklingsmetode er det viktig å få på plass en komplett kravspesifikasjon og planer for hva som skal utvikles når. Endringer er i utgangspunktet uønsket i en slik modell, og hvis endringer skal gjennomføres skjer dette gjennom en formalisert endringshåndtering. I en smidig utviklingsmetode er endringer ønsket. Det er ved nettopp å kunne justere kursen underveis at usikkerheten og kompleksiteten i et utviklingsprosjekt kan håndteres.

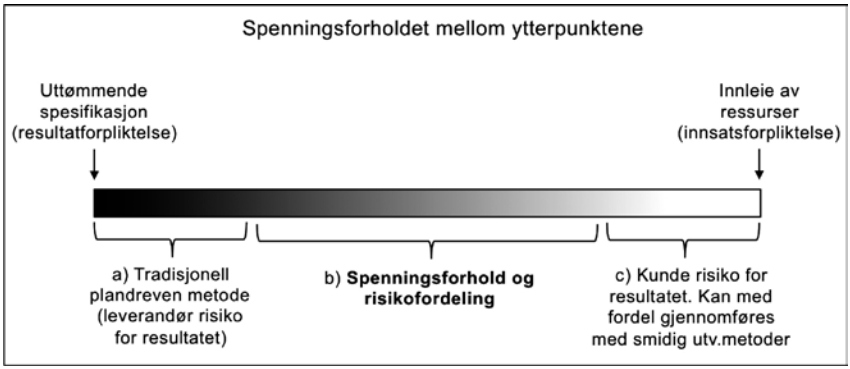
Som nevnt er kontraktsregulering av smidig programvareutvikling fortsatt under utvikling, og problemstillinger knyttet til temaet er på ingen måte gjennomdrøftet. Et overordnet *spørsmål for problemstillinger* i denne oppgaven er:

Hvilke kontraktsmekanismer kan bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes?

Spørsmålet er omfattende og uten enkle svar. Å ta sikte på en bred drøftelse av alle relevante kontraktsmekanismer innenfor dette temaet vil sprenge rammene for denne oppgaven. Jeg har derfor valgt å se nærmere på et sentralt område der det er vesentlig forskjell på en plandreven utviklingsmetode og en smidig utviklingsmetode, nemlig spesifikasjon av kontraktsgjenstanden på tidspunktet for kontraktsinngåelsen. Dette vil typisk være kravspesifikasjoner, løsningsbeskrivelser og designdokumenter. Istedenfor at programvaren er spesifisert ved kontraktsinngåelsen skal den utvikles i iterasjoner med inkrementer som skal

evalueres og avstemmes fortløpende. Endringer gjøres også fortløpende basert på evalueringen og læringen underveis i utviklingsprosessen. Dette utgangspunktet er det manifestets punkter om *programvare som virker*, *samarbeid med kunden* og *å reagere på endringer* skal understøtte. Spesifikasjon av kontraktsgjenstanden, med formaliserte endringsprosedyrer, preklusjonsregler og eventuelle sanksjoner ved oppfyllelsessvikt i henhold til spesifikasjonen, er sentrale temaer i den tradisjonelle kontraktsretten. Spenningsforholdet som oppstår mellom kontraktsregulering av en smidig utviklingsmetode og en plandreven utviklingsmetode med tradisjonell kontraktsregulering, er etter min mening kjernen i det overordnede spørsmålet.

Ved inngåelsen av en kontrakt for utvikling av programvare mellom en kunde og leverandør finnes det forenklet sagt to ytterpunkter. Det ene ytterpunktet er en tradisjonell plandreven metode med en uttømmende spesifikasjon av programvaren, hvor leverandøren har risikoen for resultatet – som en resultatforpliktelse. Det andre ytterpunktet er innleie av ressurser, som for eksempel konsulenter, hvor kunden har risikoen for resultatet og leverandøren yter en innsatsforpliktelse. Ved innleie av ressurser kan en smidig utviklingsmetode absolutt benyttes. Mange vellykkede prosjekter blir gjennomført med en slik modell. Erfaringer viser likevel at kunden ofte ikke ønsker å ta risikoen for prosjektets gjennomføring alene, og det er her det nevnte spenningsforholdet kommer inn. Figur 1.1 illustrerer forholdet.



Figur 1.1: Spenningsforholdet og risikofordelingen mellom ytterpunktene for programvareutviklingskontrakter og graden av spesifikasjon.

Som inspirasjon for drøftelsene av problemstillingen velger jeg å trekke inn såkalt *proaktiv juss*. En slik *proaktiv tilnærming* til kontrakter mener jeg er et nyttig utgangspunkt fordi den kan være et verktøy for å drøfte alternative reguleringer i lys av både kontraktsrettslige regler og de forretningsmessige verdiene som skal realiseres i et prosjekt. Proaktiv juss legger blant annet vekt på samspill mellom den juridiske profesjonen og andre profesjoner, for eksempel innen informasjonsteknologi. I underkapitlene 1.5.2 og 3.2.1 forklarer jeg nærmere om proaktiv juss og den proaktive tilnærmingen til kontrakter.

Problemstillingen som skal behandles i denne oppgaven er:
Hvordan kan alternative måter å spesifisere programvaren på bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes? Og hvordan skal dette reguleres i kontrakten med tanke på risikofordeling og kontraktsbrudd?

1.3 Videre fremstilling

Oppgavens videre fremstilling består av to deler. I den første delen forklarer jeg nærmere om programvareutviklingsmetoder. I tillegg drøfter jeg utfordringer knyttet til både programvareutvikling generelt og de kontraktsmessige utfordringene – spesielt med tanke på de tradisjonelle tilvirkningskontraktene. Dette er viktig bakgrunnskunnskap for oppgavens neste del, som er drøftelsen av kontraktsregulering av smidig programvareutvikling med alternative måter å spesifisere programvaren på.

Før jeg går løs på første del er det nødvendig med noen presiseringer av viktige begreper som brukes i oppgaven, samt noen avgrensninger mot prosjekt- og kontraktstyper som ligger utenfor oppgavens problemstilling. I tillegg gjør jeg en avgrensning mot temaer som ligger tett opp til oppgaven, men som ikke skal behandles. Videre gjør jeg rede for oppgavens metode.

1.4 Presiseringer og avgrensninger

1.4.1 Begrepet *programvareutvikling*

Begrepet *programvareutvikling* blir gjennomgående brukt i oppgaven og det er nødvendig å avklare hva som menes. Et it-prosjekt kan bestå av mange aktiviteter som for eksempel planlegging, økonomistyring, anbudshåndtering, behovsanalyse, kravspesifikasjon, programmering, testing, opplæring, installasjon, drift og vedlikehold. *Programvareutvikling* kan defineres i både vid og snever forstand. I vid forstand blir også betegnelsen *systemutviling* brukt, og på engelsk tilsvarer dette «software engineering». Når den vide betydningen brukes hand-

ler det om fire aktiviteter: programvarespesifikasjon, programvareutvikling, programvarevalidering og programvareevolusjon.[15] Jeg kommer nærmere tilbake til beskrivelsen av prosessen for utvikling av programvare, men følgende beskriver de fire aktivitetene i korte trekk: programvarespesifikasjon er en beskrivelse av hva som skal lages, eventuelle tekniske rammer og krav, samt hvilke behov brukerne har. Basert på spesifikasjonen produseres og utvikles programvaren – altså programvareutviklingen. Underveis i prosessen avstemmes det som utvikles med brukerne og kravene (programvarevalidering). Produktet må ofte endres og justeres underveis basert på valideringen – dette kalles for programvareevolusjon. Den av de fire aktivitetene som er selve programmeringsjobben kalles også *programvareutvikling*. Dette er begrepet i snever forstand, og på engelsk brukes uttrykket «software development». Etter min oppfatning virker det som om betegnelsen *programvareutvikling* i vid betydning brukes hyppigere enn i snever betydning. Jeg har derfor valgt å bruke begrepet i vid betydning i denne oppgaven.

1.4.2 Avgrensning av prosjekttyper og kontraktstyper

Det finnes mange ulike kontraktstyper og it-prosjekter. Generelt kan et prosjekt defineres som en midlertidig og tidsavgrenset bestrebelse for å skape et produkt, tjeneste eller resultat.[16] I denne oppgaven handler det om it-prosjekter[17] av en viss størrelse hvor hovedaktiviteten er utvikling av programvare spesialtilpasset brukere i en organisasjon, jeg bruker også betegnelsene *system* eller *løsning*.[18] Slik programvare kan være bygget opp fra bunnen av, eller ved større eller mindre grad av gjenbruk av eksisterende programvarekomponenter. Motstykket til spesialtilpasset programvare er såkalt standard programvare, som også kalles «hyllevare». Eksempler på dette er Microsoft Office, operativsystemer som Windows, OSx og en rekke applikasjoner til smarttelefoner og nettbrett. Jeg avgrenser videre mot interne prosjekter – og tar bare for meg prosjekter med kunde-leverandør-forhold. Det typiske avtaleforholdet vil da være mellom en profesjonell programvareleverandør og en kunde som ikke er programvareleverandør. Forbrukerforhold holdes utenfor.

Den kontraktstypen som kan regulere et programvareutviklingsprosjekt, som definert over, er en *tilvirkningskontrakt*. Lasse Simonsen definerer en tilvirkningskontrakt som følger: «Med tilvirkningskontrakter menes i alminnelighet avtaler hvor realdebitor […] påtar seg å fremstille et nærmere bestemt produkt for realkreditor […], det være seg en løsøregjenstand (borerigg, maskin, skip,

15 Jf. Sommerville (2011) s. 9

16 Se PMI (2013) s. 3

17 Informasjonsteknologi-prosjekter

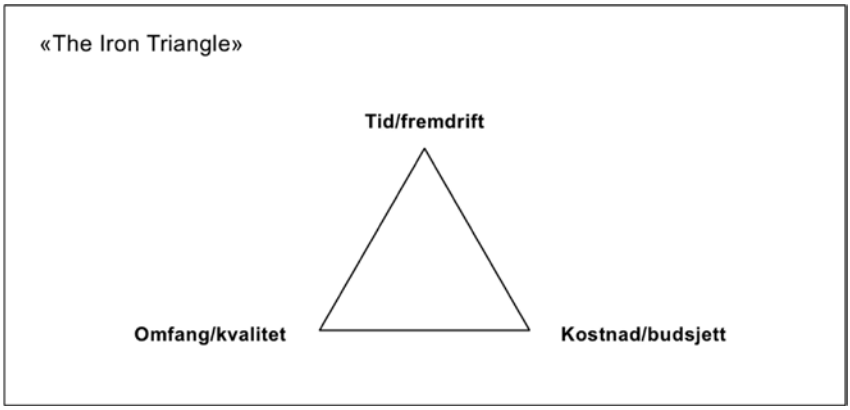
18 En generell definisjon av *system* i denne sammenheng har Sommerville (2011) s. 266: «A system is a purposeful collection of interrelated components, of different kind, which work together to achieve som objective.»

mv.) et bygg eller anlegg, eller et rent åndsprodukt (så som arkitekttegninger og dataprogrammer)»[19]. Selve tilvirkningsprosessen preger kontraktene i større grad enn det ferdige produktet. Det er vanskelig å definere hvor grensene for en tilvirkningskontrakt går – spesielt når det gjelder avtalene om innleie av konsulenter. Figur 1.1 kan også brukes her som en illustrasjon på ytterpunktene. Til venstre i figuren ligger de typiske tilvirkningskontraktene, mens til høyre ligger typisk bistandsavtaler hvor det er selve arbeidsinnsatsen som reguleres. Definisjonen vil etter dette være noe upresis, men for oppgavens del er det tilstrekkelig å avgrense mot kontrakter som gjelder kun innleie av konsulenter.

1.4.3 Hva betyr det å *lykkes* med et it-prosjekt?

Denne oppgaven handler om å lykkes med smidig programvareutvikling i it-prosjekter. Det er derfor nødvendig å klargjøre det betyr å *lykkes* med et it-prosjekt. Om et prosjekt er vellykket vurderes ut fra om visse *suksesskriterier* er oppfylt.[20] Når en slik vurdering skal gjøres er det viktig å få frem at et prosjekt ikke enten er vellykket eller mislykket – men at prosjektene kan være vellykkede i varierende grad.[21] Tradisjonelt har prosjektene blitt vurdert med tanke på tre suksesskriterier; *kostnad/budsjett*, *tid/fremdrift* og *omfang/kvalitet*. Disse kriteriene henger tett sammen med tradisjonelle plandrevne metoder hvor kostnad, fremdrift og omfang er faste størrelser ved kontraktsinngåelsen. Derfor måles prosjektet på om gjennomføringen holder seg innenfor budsjett og definert sluttdato. I tillegg defineres kvaliteten på løsningen ut ifra om den har spesifiserte egenskaper. Disse faste suksesskriteriene omtales på engelsk som «The Iron Triangle», og illustreres som i figur 1.2. I tillegg til at kriteriene kostnad, tid og kvalitet henger sammen med bruk av plandrevne metoder, er de også tett knyttet til vanlige definisjoner av selve *prosjektstyringen*. En prosjektleder har mange oppgaver, men sentrale oppgaver har nettopp vært å sørge for at prosjektgjennomføringen holder seg innenfor «The Iron Triangle».[22]

19 Se Simonsen (1999) s. 307
20 Jf. Karlsen (2013) s. 488
21 Se Wateridge (1998) s. 59
22 Se Atkinson (1999) s. 337-338

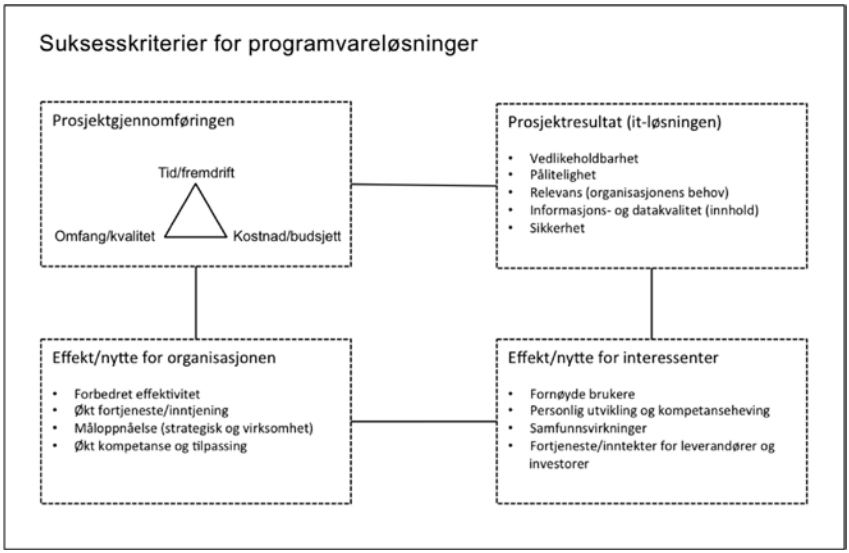


Figur 1.2: «The Iron Triangle» som tradisjonelle suksesskriterier

Det finnes mange prosjekter som blir betegnet som vellykkede selv om ikke de tradisjonelle suksesskritene er oppfylt. Det finnes også mange prosjekter, som til tross for at de oppfyller disse suksesskritene, regnes som fiaskoer. Derfor har det utviklet seg enighet om at vurderingen av om et prosjekt er vellykket må baseres på flere kriterier.[23] I tillegg til å vurdere om prosjektgjennomføringen holder seg innenfor definerte rammer, er det vel så viktig å vurdere om prosjektet for eksempel gir ønskede effekter og resultater etter at prosjektet er avsluttet. Det kan være om brukerne er fornøyde med løsningen, om den gir ønskede organisatoriske effekter og om leverandørene har hatt fortjeneste. Mange av disse ønskede effektene dreier seg om såkalt *gevinstrealisering*.[24] Gevinstrealisering er et viktig element i både smidig programvareutvikling og kontraktsutforming med proaktiv juss. Jeg kommer nærmere tilbake til gevinstrealisering i underkapittel 3.4.1.

Det er gjort en del forskning på å definere ulike typer suksesskriterier som bør ligge til grunn for vurderingen av i hvilken grad et prosjekt er vellykket eller ikke. Jeg tar utgangspunkt i Atkinsons tilnærming,[25] i tillegg til Karlsens oversikt.[26] Den siste bygger delvis på Atkinson, men også på en undersøkelse gjort av Karlsen og Gottschalk. Figur 1.3 gir oversikt over suksesskriterier, med eksempler på delkriterier, som bør ligge til grunn for vurderingen.

23 Jf. Karlsen (2013) s. 489
24 Jf. Karlsen (2013) s. 490
25 Atkinson (1999)
26 Karlsen (2013) s. 490-491



Figur 1.3: Et utvidet syn på suksesskriterier for it-prosjekter

Figur 1.3 viser at suksesskriterier for it-prosjekter er utvidet til fire dimensjoner. De tradisjonelle suksesskriteriene i «The Iron Triangle» er fortsatt med, og er kun knyttet til selve *prosjektgjennomføringen*. I noen prosjekter vil kanskje de tradisjonelle kriteriene ha større vekt enn øvrige kriterier. Den andre dimensjonen er knyttet til selve løsningen som utvikles – *prosjektresultatet*. Disse kriteriene kan også regnes som kvaliteten på løsningen etter at den er satt i produksjon. I noen versjoner av «The Iron Triangle» er denne kvaliteten tatt inn som et fjerde suksesskriterium. De to øvrige dimensjonene handler om effekt og nytte for henholdsvis organisatoriske forhold og overfor øvrige interessenter. Noen av disse kriteriene er ikke mulig å vurdere før etter at prosjektet er avsluttet, og noen så sent som flere år etter at systemet er satt i produksjon.[27] Det blir mer og mer problematisk å vurdere oppfyllelsesgraden av suksesskriterier etter hvert som tiden går, fordi ulike forhold i for eksempel organisasjonen påvirker vurderingen.[28] Ved bruk av smidig programvareutvikling med hyppige leveranser av inkrementer kan visse suksesskriterier knyttet til effekt og nytte måles og vurderes underveis.[29]

27 Jf. Atkinson (1999) s. 340

28 Jf. Karlsen (2013) s. 494

29 Se Atkinson (1999) s. 340 som relaterer dette til en iterativ utviklingsmetode (RAD).

Hvilke suksesskriterier som skal legges til grunn i det enkelte prosjekt vil variere, fordi det ligger i prosjektets natur at de alle er unike. I utgangspunktet bør definisjonen av suksesskriterier for et prosjekt foreligge før prosjektet starter. Likevel kan det være nødvendig å justere disse underveis i prosjektet.[30] Suksesskriteriene er viktige når det gjelder vurderingen av prosjektet, men også som styrende for gjennomføringen. Ikke minst for prosjektmedlemmene og øvrige interessenter er kriteriene viktig med tanke på forventningsavstemming. På bakgrunn av dette ser jeg det som nødvendig at suksesskriteriene tas med i kontrakter for programvareutvikling.

1.4.4 Øvrige avgrensninger

Offentlig anskaffelse av programvare som skal utvikles ved bruk av smidige metoder kan medføre noen utfordringer med tanke på anskaffelsesregelverket. Dette gjelder spesielt i anbudsprosessen, men er også relatert til endringer i prosjektet etter kontraktsinngåelsen og tolkning av kontrakten.[31] Den vanlige fremgangsmåten er at kunden utarbeider en kravspesifikasjon som er grunnlaget for leverandørenes tilbud. En slik fremgangsmåte kan passe hvis prosjektet skal gjennomføres med en plandreven utviklingsmetode som forutsetter at kravspesifikasjonen er uttømmende. Hvis en slik kravspesifikasjon ikke kan legges til grunn i anbudsprosessen vil andre tildelingskriterier få større betydning. I tillegg ville det være ønskelig med dialog mellom kunden og tilbyderne, for å avklare hvordan kundens behov kan løses. Et alternativ kan være såkalt *konkurransepreget dialog*.[32] Likevel er det synspunkter på at de gjeldende anskaffelsesreglene ikke er tilpasset alternative anskaffelsesprosesser, eller at regelverket ikke utnyttes fullt ut, som gjør innovasjon i offentlig sektor vanskelig.[33] Dette er viktige og interessante problemstillinger, men ligger utenfor rammene i denne oppgaven.

1.5 Metode

Utformingen av kontrakter skal ivareta flere, ofte motstridende, hensyn. En oversiktlig, enkel og fleksibel utforming kan lett komme i konflikt med et ønske om tydelig og klar regulering av alle vesentlige spørsmål.[34] I prosjekter med smidig programvareutvikling er behovene for oversiktlige, enkle og fleksible kontrakter sterkere enn i de mer tradisjonelle prosjektene. Det er også nødvendig å vurdere kontraktens utforming med tanke på at den skal fungere som et verktøy

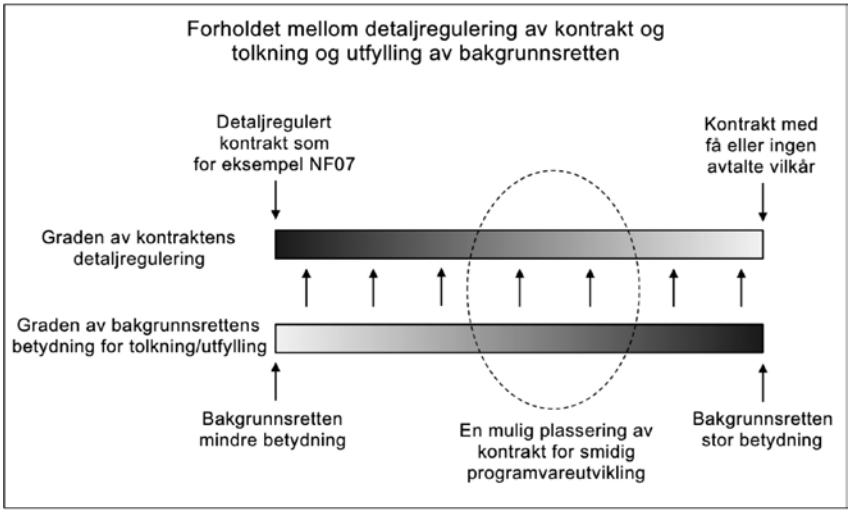
30 Se nærmere Karlsen (2013) s. 492-494
31 Se Krüger (2010)
32 Se Forskrift om offentlige anskaffelser (2006) §§ 20-8 og 20-9
33 Se for eksempel kronikk Hoff (2014)
34 Torvund (1997) s. 48

for styring av utviklingsmetoden og bidra til utstrakt samarbeid mellom partene. I tillegg må kontrakten ivareta partenes forretningsmål. Det gir noen metodiske utfordringer å drøfte kontrakter som, i større grad enn tradisjonelle kontrakter, skal være en katalysator for samarbeid, fleksibilitet og måloppnåelse. Derfor velger jeg å supplere den juridiske metoden på kontraktsrettens område. Det gjør jeg ved å trekke inn forskning, litteratur og erfaringsgrunnlag fra profesjoner innen informasjonsteknologi og prosjektstyring. For å få til drøftelser i et perspektiv som er noe bredere enn en juridisk metode, bruker jeg som nevnt *proaktiv juss* som en innfallsvinkel.

1.5.1 Rettskildebildet

På kontraktsrettens område er det avtalen i seg selv om er utgangpunktet for hvordan forholdet mellom partene skal reguleres. For å fastlegge kontraktens innhold må både ordlyd, indre systematikk og partenes felles oppfatning tolkes. I tillegg skal vilkårene «tolkes slik at det tilstrebes et resultat som fremtrer som *rimelig og fornuftig*»[35]. Hvis kontraktens innhold ikke kan fastlegges basert på tolkning som tar utgangspunkt i selve kontrakten, må kontrakten tolkes og utfylles med deklaratorisk bakgrunnsrett. Jo mer uttømmende detaljregulert kontrakten er, jo mindre betydning får bakgrunnsretten for tolkning og utfylling. Kontraktsregulering av smidig programvareutvikling vil som nevnt skille seg noe fra mer tradisjonelle tilvirkningskontrakter – spesielt med tanke på fleksibilitet og enkelhet. Dette gjør nok at bakgrunnsretten får større betydning for disse kontraktene enn en omfattende detaljregulert kontrakt som for eksempel fabrikasjonskontrakten NF 07. Figur 1.4 illustrerer dette forholdet.

35 Hagstrøm (2011) s. 43



Figur 1.4: Forholdet mellom kontrakters detaljregulering og bakgrunnsrettens betydning, samt plassering av kontrakt for smidig programvareutvikling.

Bakgrunnsretten består av prinsipper og regler som er utviklet gjennom rettspraksis og kontraktspraksis, og gjelder som utgangspunkt for alle typer kontrakter. Kjøpsloven[36] er i stor grad en kodifisering av den ulovfestede bakgrunnsretten. Det antas likevel at kjøpslovens regler om tilvirkningskjøp ikke får direkte anvendelse på programvareutviklingskontrakter, men at den kan være en mulig kilde ved tolkning og utfylling.[37] Programvareutviklingskontrakter ligger nok nærmere tjenesteyting og entreprise enn et tilvirkningskjøp etter kjøpslovens regler,[38] og kontraktene kan settes i samme bås som entreprise- og fabrikasjonskontrakter.[39] Dette gjør at tilgangen på relevante rettskilder øker betraktelig sammenliknet med det sparsomme materiale som finnes på området for programvareutviklingskontrakter. De ulovfestede reglene på entrepriserettens område kan få betydning for drøftelser av programvareutviklingskontrakter. Det er videre utviklet godt etablerte standardkontrakter for både entreprise og fabrikasjon. Det er også laget standardkontrakter for programvareutvikling. Siden standardkontrakter har stor betydning på disse områdene, og at kontraktspraksis er en rettskilde, redegjør jeg nærmere for hvilke standardkontrakter som er relevante i denne sammenheng.

36 Kjøpsloven (1988)
37 Føyen (2006) s. 57
38 l.c. og Torvund (1997) s. 24
39 Se nærmere avgrensning i underkapittel 1.4.2

1.5.1.1 Standardkontrakter

Standardkontrakter kan forenkle kontraktsinngåelsen og transaksjonskostnadene reduseres.[40] Disse kontraktene er gjennomarbeidet for å tydeliggjøre eventuell gjeldende bakgrunnsrett, eller fravike deklaratorisk bakgrunnsrett. Dette kan for eksempel være ansvarsbegrensninger og formaliserte varslingsregler.[41] Standardkontrakter på entrepriserettens område har lang historie og omfattende bruk. I denne sammenheng er de gjeldende *Norsk bygge- og anleggskontrakt* (NS 8405)[42] og *Alminnelige kontraktsbestemmelser for totalentrepriser* (NS 8407)[43] som er aktuelle. I petroleumsvirksomheten er det utviklet fabrikasjonskontrakter som har relevans for programvareutviklingskontraktene. Aktuelle standardkontrakter på dette område er *Norsk Fabrikasjonskontrakt* (NF 07)[44] og *Norsk Totalkontrakt* (NTK 07)[45]. Både standardkontraktene for entreprise og fabrikasjon er utarbeidet ved forhandlinger mellom leverandør- og kundesiden, og er såkalte «agreed documents». Dette betyr at disse standardkontraktene har større vekt som rettskilde på allment grunnlag, når standardvilkårene ikke er avtalt mellom partene.[46]

Det er også utarbeidet flere standardkontrakter for programvareutvikling med ensidig opprettede standardvilkår. Dette gjør at disse ikke har den samme vekten som rettskilde for kontrakter som ikke eksplisitt har benyttet standardvilkårene. Likevel kan disse kontraktene være en kilde for argumenter til løsning av spørsmål, spesielt siden disse standardkontraktene har hentet inspirasjon fra fabrikasjonskontraktene, og særlig den tidligere NF 92.[47] Et annet poeng er at programvareutviklingsprosjekter som drøftes i denne oppgaven ofte vil benytte en av de eksisterende standardavtalene som finnes på markedet.

Standardkontrakter for programvareutvikling er utarbeidet av Difi, Den Norske Dataforening (DND) og IKT-Norge. Difi utvikler og forvalter statens standardavtaler, og for programvareutvikling er de relevante avtalene *Programutviklingsavtalen* (SSA-U)[48] og *Smidigavtalen* (SSA-S)[49]. SSA-U tar utgangspunkt i en tradisjonell plandreven utviklingsmetode, mens den ferske avtalen SSA-S, som ble lansert i januar i 2014, kan benyttes for smidige utviklingsmetoder. DND tilbyr *PS2000 Standard*[50], *PS2000 Smidig*[51], og *PS2000 Smidige Oppdrags-*

40 Se Eide (2008) s. 289
41 Jf. Hagstrøm (2011) s. 46 (petit)
42 Standard Norge (2008)
43 Standard Norge (2011)
44 Norsk Industri (2007a)
45 Norsk Industri (2007b)
46 Hagstrøm (2011) s. 60
47 Kaasen (2006) s. 48
48 Difi (2013)
49 Difi (2014)
50 Den Norske Dataforening (2010b)
51 Den Norske Dataforening (2010a)

baserte Leveranser (PS2000 SOL)[52]. PS2000 Standard ble lansert i 2000 og var da den eneste programvareutviklingsavtalen som la opp til en iterativ utviklingsmetode. Den har blir brukt i større prosjekter med vellykkede resultater.[53] PS2000 Smidig bygger på PS2000 men med et bilagssett og veiledning som skal være tilpasset smidige utviklingsmetoder. Både PS2000 Standard og PS2000 Smidig legger opp til en mer eller mindre utarbeidet spesifikasjon ved kontraktsinngåelsen. Den nye PS2000 SOL som ble lansert i 2013 skal ha som utgangspunkt at omfanget ikke skal defineres ved kontraktsinngåelsen, og at avtalen skal ha mer preg av en konsulentbistandsavtale. IKT-Norges avtale *Systemutviklingsprosjekt – Standardavtale* (IKT 2010)[54] har mange likheter med SSA-U og baserer seg også på en tradisjonell plandreven utviklingsmetode. IKT-Norge har også en avtale for mindre programvareutviklingsprosjekter.

Ved drøftelse av kontrakter for smidig programvareutvikling er det interessant å trekke inn såkalte *samspillkontrakter*. Disse kalles også partneringkontrakter eller prosjektalliansekontrakter. Det spesielle ved en samspillkontrakt er at den « […] legger til grunn en særlig samhandling mellom partene som er basert på åpenhet og tillit.»[55] Samarbeid og tillit er viktige elementer ved en smidig utviklingsmetode. Derfor er utformingen av slike samspillkontrakter av interesse. På fabrikasjonskontraktenes område ble det utarbeidet en slik type kontrakt: *Norsk Prosjektalliansekontrakt 1996 (NPA 96).* Denne standardkontrakten ble imidlertid aldri tatt i bruk.[56] Innenfor entrepriseretten har slike kontrakter fått et visst fotfeste, og det er gjennomført flere prosjekter med denne kontraktsvarianten som har vært vellykkede.[57] Bergsaker trekker frem visse behov i et prosjekt som gjør slike kontrakter fordelaktige: komplekse prosjekter, behov for særlig kompetanse og fleksibilitet med tanke på ressurser.[58] Etter min mening er slike behov fremtredende i et smidig utviklingsprosjekt. Jeg nevner også at Standard Norge har utredet behovet for reviderte standardkontrakter med tanke på blant annet at antallet samspillprosjekter har et visst omfang.[59] Entreprenørforeningen – Bygg og Anlegg (EBL) har utarbeidet en *Veileder om SAMSPILL-ENTREPRISE*.[60] Både denne veilederen og Standard Norges rapport er interessante kilder til inspirasjon og argumenter i oppgavens drøftelse.

52 Den Norske Dataforening (2013a)
53 Se for eksempel Statens pensjonskasse (2012) s. 34
54 IKT-Norge (2010)
55 Bergsaker (2010) s. 170
56 Kaasen (2006) s. 26-27 (petit)
57 Bergsaker (2010) s. 171
58 l.c.
59 Standard Norges komité SN/K 534 (2013)
60 Entreprenørforeningen – Bygg og Anlegg (2013)

1.5.1.2 Juridisk teori

Spesielt for spørsmål som ikke er løst i andre rettskilder er teorien en viktig kilde.[61] På kontraktsrettens område for programvareutvikling er det få øvrige rettskilder. Føyen mfl. har en samlet behandling av kontrakter for programvareutvikling i *Kontrakter for utvikling av programvare*.[62] Boken legger størst vekt på en plandreven utviklingsmetode, men tar også for seg iterative metoder. Torvunds *Kontraktsregulering – IT kontrakter* drøfter først og fremst programvareutvikling med en plandreven utviklingsmetode.[63] Kontraktsretten i Norden er utviklet i et samarbeid, og teori fra disse landene er også relevant.

1.5.1.3 Reelle hensyn

Når det gjelder kontrakter for smidig programvareutvikling finnes det svært lite rettskildemateriale, og oppgaven drøfter kontraktsmekanismer som er lite behandlet i litteraturen. Dette gjør at argumenter og momenter som bunner i reelle hensyn vil få større vekt. Kravet til lojalitet i kontraktsforhold er blitt strengere og viktigere de siste årene.[64] Dette vil nok også gjelde enda tydeligere for kontraktstyper som baserer seg på samarbeid og tillit. Rimelighetsvurderinger, også basert på de lege ferenda-betraktninger, kan derfor være en viktig del av drøftelsene. I tillegg kan det være interessant å trekke inn rettsøkonomiske betraktninger som reelle hensyn når kontraktsmekanismer skal behandles. Omfattende kontraktsregulering av komplekse tilvirkningskontrakter kan resultere i høye transaksjonskostnader. Hvis samspillskontrakter trekkes inn som en rettskilde kan også rettsøkonomiske aspekter bidra, fordi et av partenes mål i disse avtalene er en felles økonomisk nytteoptimering.[65]

1.5.2 Proaktiv juss

Proaktiv juss, som på engelsk betegnes som «proactive law», er en juridisk tilnærmingsmåte som tar sikte på å løse rettslige spørsmål *ex ante*. Det vil si at de rettslige spørsmålene skal løses *før* en hendelse oppstår. En hendelse kan i denne sammenheng både være et problem eller en mulighet. Derfor vektlegger proaktiv juss å realisere potensielle muligheter som kan gi forretningsmessig gevinst. I følge Dag Wiese Schartum handler proaktivitet i denne sammenheng om «[…] addressing and – as far as possible – eliminating uncertainty, both in the sense of avoiding risks and realising opportunities».[66] Proaktiv juss har sitt utspring i Finland fra slutten av 1990-tallet. Helena Haapios arbeid med det proaktive perspektivet på kontraktsrettens område har vært utgangspunktet for

61 Jf. Hagstrøm (2011) s. 82
62 Se Føyen (2006)
63 Se Torvund (1997)
64 Se Hagstrøm (2011) s. 77
65 Tvarnø (2002b) s. 80
66 Schartum (2006) s. 38

den proaktive jussen. Proaktiv juss blir stadig videreutviklet og tilnærmingen får fotfeste innen flere rettsområder og i flere rettstradisjoner utover Norden.[67]

Den proaktive tankegangen og *ex ante*-perspektivet finnes også i den såkalte *preventive juss* som ble introdusert i 1950 av Lous M. Brown.[68] Utgangspunktet til preventiv juss er at det koster mindre å unngå problemer enn å komme seg ut av problemene når de først har oppstått. En slik tankegang ligger bak de velutviklede metodene for alternativ tvisteløsning i blant annet USA. Den tankegangen ligger også til grunn for at tvisteloven[69] legger vekt på at partene skal forsøke å oppnå en minnelig løsning. Kontrakten har også en preventiv funksjon ved at den først og fremst skal være konfliktforebyggende, og den skal bidra til enklere problemløsning når konflikten har oppstått.[70] Selv om kontrakten til dels har en ex ante-funksjon er likevel det tradisjonelle utgangspunktet at det er *ex post*-funksjonen som er fremtredende. Behandlingen av avtaler i kontraktsretten konsentrerer seg mye om hvordan kontrakten og bakgrunnsretten skal løse problemer som allerede har oppstått.[71] Dette reaktive utgangspunktet preger naturligvis kontraktsvilkårene, og det er den juridiske profesjonens (dommerens) tilnærming som ligger til grunn for utformingen.

En *proaktiv tilnærming* til kontrakten innebærer å utvide den fra å være et «[…] overordnet styringsdokument som sier hva partene har påtatt seg å gjøre»[72] til et styringsverktøy som i større grad skal ivareta partenes forretningsmessige mål. Jeg siterer Helena Haapios sammenfatning om at den proaktive tilnærmingen «[…] seeks to enable the parties to achieve the objectives of their collaboration: the parties can use contracts, first and foremost, to frame, support and guide successful business deals and relationships, with an emphasis on good communication and collaboration; second, to balance risk with reward; and, third, to prevent unnecessary problems, disputes and litigation.»[73] Verdiene og prinsippene som ligger bak en smidig programvareutviklingsmetode har interessante likhetstrekk som gjør at denne proaktive tilnærmingen kan gi viktige bidrag til kontraktsutformingen. Henschel trekker frem at en proaktiv tilnærming til utformingen fordrer at kontraktsmekanismene klart henger sammen med faktorer som bidrar at partene oppnår forretningsmessig verdi.[74] Han konkluderer med at kontrakter tilpasset smidig programvareutvikling i seg er et verktøy basert på en proaktiv tilnærming.[75]

67 Berger-Walliser (2012) s. 23-25 med fotnoter
68 ibid. s. 20
69 Tvisteloven (2005)
70 Jf. Torvund (1997) s. 42
71 Se Haapio (2013) s. 2
72 Jf. Torvund (1997) s. 42
73 Se Haapio (2013) s. 7
74 Henschel (2012) s. 239
75 ibid. s. 249-250

2 Programvareutvikling og kontraktsforhold

Når kontraktsregulering av programvareutvikling skal drøftes er det nødvendig å ha kunnskap om hva som er spesielt med programvareutvikling, samt hvordan programvareutviklingen organiseres og gjennomføres i praksis. Derfor forklarer jeg nærmere i kapittel 2.1 hvilke særtrekk programvareutvikling har. I kapittel 2.2 til 2.4 presenterer jeg ulike måter å organisere programvareutviklingen på. Til slutt, i kapittel 2.5, går jeg over til å drøfte noen kontraktsmessige spørsmål knyttet til programvareutvikling, endringshåndtering og risikofordeling.

2.1 Særtrekk ved programvareutvikling

Programvareutvikling har flere likheter med andre tilvirkningsprosjekter. Dette gjør at mange kontraktsrettslige spørsmål kan drøftes på tvers av for eksempel programvareutvikling, entreprise og fabrikasjon. Selv om programvareutvikling ikke er plaget med vær- og grunnforhold er det en del andre særtrekk som gjør at disse prosjektene likevel skiller seg fra prosjekter i andre bransjer. Disse særtrekkene vil kunne påvirke hvordan utviklingsmetodene bør utformes, og hvilken metode som bør velges i de ulike prosjektene. Særtrekkene vil også få betydning for kontraktsutformingen. Særlig med hensyn til spesifikasjon av det som skal tilvirkes, endringshåndteringen og ikke minst risikofordelingen mellom kunde og leverandør.

2.1.1 Kompleksiteten

Tilvirkning av programvare er en kompleks affære. Det finnes alltid en rekke måter å løse de ulike oppgavene på – kanskje like mange måter som det finnes utviklere. I tillegg består programvaren av utallige instruksjoner og komponenter som innbyrdes skal henge sammen. Selv ved små endringer i programkoden er det vanskelig å vite hvilke konsekvenser endringene har. Sannsynligheten for feil øker eksponentielt med antall instruksjoner og antall komponenter, og dermed øker den tekniske kompleksiteten drastisk.[76] Denne kompleksiteten gjorde seg gjeldende tidlig i programvareutviklingens historie. Derfor ble konseptet med såkalt *objektorientert programmering* utviklet allerede i 1964 av Ole-Johan Dahl og Kristen Nygaard ved Norsk Regnesentral. Dette konseptet innebærer å gruppere programkode i autonome enheter (objekter) som logisk hører sammen.

76 Jf. Brooks Jr. (1995) s. 183

Konseptet ble etter hvert bygget inn i de fleste moderne programmeringsspråk, og er fortsatt et svært viktig verktøy for håndtering av kompleksitet. Til tross for denne måten å programmere på minsker ikke kompleksiteten i programvaren. Kravene til nye programvaresystemer øker stadig og det er lite som tyder på at omfanget og kompleksiteten avtar med tiden. I tillegg gjør alle integrasjonene mot andre systemer utfordringene enda større. I løpet av de siste tretti årene har utviklingen gått fra mindre enbrukersystem til store flerbrukersystemer med mange integrasjoner. Følgende er en god beskrivelse av programvarens kompleksitet:

> «Computer programs are the most intricate, delicately balanced and finely interwoven of all the products of human industry to date. They are machines with far more moving parts than any engine: the parts don't wear out, but they interact and rub up against one another in ways the programmers themselves cannot predict.»[77]

2.1.2 Sosiotekniske systemer

Spesialtilpassede systemer skal ofte samhandle med mennesker og innarbeides i organisasjonen. Slike systemer kan karakteriseres som såkalte *sosiotekniske systemer*. Innføringen av nye systemer kan få store konsekvenser for arbeidsprosesser, ansettelser og organisasjonsendringer.[78] Dette skaper store utfordringer for leveransen – både når det gjelder selve utviklingen og påfølgende bruk. En av årsakene til at it-prosjekter mislykkes er nettopp at det ikke er tatt hensyn til det sosiotekniske perspektivet, og ønskede effekter etter avsluttet prosjekt har uteblitt. Her skiller programvareutviklingen seg mye fra for eksempel entreprise eller fabrikasjon, fordi et bygg eller en borerigg tilvirkes isolert fra kundens organisasjon.[79]

2.1.3 Abstrakte og usynlige systemer

Et aspekt som skiller programvareutvikling fra for eksempel entreprise, er det abstrakte kontra det håndfaste. Et byggverk kan i stor grad visualiseres med tegninger og modeller før byggeprosessen starter. Ikke minst kan man ta det faktiske bygget som skrider frem nærmere i øyensyn. Dette skiller seg radikalt fra programvare. Det er svært vanskelig å utforme tegninger og beskrivelser som gjør at utviklere og brukere får et realistisk og riktig bilde av hva som faktisk skal lages.[80] En konsekvens av dette er at det også blir svært vanskelig å lage en spesifikasjon av programvaren. Spesielt for kunden, som ikke har kunnskapen om hvilke muligheter og begrensninger som finnes, er dette vanskelig.

---

77 Gleick (1992) s. 2

78 Se nærmere Sommerville (2011) s. 267-268

79 Jf. Torvund (1997) s. 27

80 Se Stepanek (2005) s. 10

I tillegg til abstraksjonsproblemet kan programvaren sees på som et isfjell. Det som er synlig for andre enn utviklerne er en svært liten del av systemet. Det meste av programkoden er ikke knyttet til det såkalte *grafiske brukergrensesnittet* (GUI).[81] Dette gjør det vanskelig å følge med på fremdriften, noe som kan skape usikkerhet hos kunden. Det er en annen situasjon for en byggherre som kan følge med på fremdriften i et byggeprosjekt.

2.1.4 Mangel på modenhet

Kunnskap og metoder for konstruksjon og oppføring av bygninger og infrastruktur har eksistert i flere tusen år. De første standardiserte kontraktsvilkårene for entreprise i Norge ble utarbeidet allerede i 1890.[82] Programvareutvikling er en ung ingeniørdisiplin. Det var først på 1980-tallet utbredelsen av maskinvare skjøt fart blant privatpersoner og i bedrifter. Behovet for programvare økte, og nye programmeringsspråk og programmeringsteknikker så dagens lys. Den nevnte objektorienterte programmeringen fikk også fotfeste. Det pågår fortsatt et kontinuerlig arbeid med å lage nye programmeringsspråk, metoder og konsepter for utvikling av programvare. Dette gjør at det skorter på modenhet hos både kunder og leverandører, og bransjen mangler fortsatt etablerte *best practices* for hvordan ulike oppgaver skal løses.[83]

2.1.5 Konstruksjon og forskning

Entreprise kan deles opp i to ulike aktiviteter: prosjektering og utførelse. Utførelsen er den fysiske arbeidsinnsatsen for å oppføre byggverket, mens prosjekteringen er en intellektuell aktivitet.[84] Denne oppdelingen kan ikke gjøres for programvareutvikling. Både detaljspesifikasjon, design og selve programmeringen kan utgjøre både prosjektering, konstruksjon og utførelse. Det er ingeniører og informatikere som står for disse aktivitetene, og all programvareutvikling er intellektuelt arbeid.

It-prosjektene må også håndtere den raske teknologiutviklingen og umodenheten i bransjen. Dette vil ofte føre til at programvareutviklingen bærer preg av forskning og utprøving. Læring underveis blir viktig for å finne nye måter å løse oppgavene på.[85] Disse forholdene kan få betydning for både risikohåndteringen og risikofordelingen mellom kunde og leverandør.

---

81 *Grafisk brukergrensesnitt* er bindeleddet mellom brukerne og systemet, altså de skjermbildene brukerne forholder seg til når de benytter systemet.

82 Se Sandvik (1966) s. 61

83 Se nærmere Stepanek (2005) s. 13-14

84 Jf. Torvund (1997) s. 28

85 Se Stepanek (2005) s. 16-17

2.2 Generelt om programvareutvikling

Uansett hvilke type prosjekt som etableres for å utvikle et produkt, er det noen aktiviteter som må gjennomføres som et minimum. Alle metoder for programvareutvikling må minst inneholde følgende aktiviteter: spesifikasjon, programmering, validering og evolusjon. Før jeg går nærmere inn på de ulike utviklingsmetodene gir jeg en oversikt over disse fire aktivitetene.

Spesifikasjon av hva som skal programmeres i et prosjekt er nødvendig. Spesifikasjonen kan bestå av tegninger, modeller og beskrivelser, og det er kundens behov som er utgangspunktet for spesifikasjonene. Kundens behov kan være alt fra brukeres oppgaveløsning til organisatoriske mål og fremtidig verdiskapning. Behovene analyseres og omsettes som regel til detaljerte beskrivelser av såkalte *funksjonelle krav* og *ikke-funksjonelle krav*. Funksjonelle krav er typisk knyttet til brukernes oppgaver som skal løses ved hjelp av systemet. Ikke-funksjonelle krav er underliggende krav til systemet som for eksempel sikkerhet, ytelse og kapasitet. I tillegg til tekstlige beskrivelser av behov og krav, er det laget ulike konsepter for grafisk representasjon av noen type krav.

Programmeringen innebærer blant annet å skrive instruksjoner (koding) som får programvaren til å virke etter spesifikasjonene. Ofte består et utviklingsprosjekt av en eller flere team hvor de fleste medlemmene utfører koding. I tillegg jobber de med design av den tekniske arkitekturen, databaser og brukergrensesnitt. Samtidig med kodingen gjør den enkelte programmereren også testing og feilsøking i egen og andres koding, såkalt *debugging*.

Validering er ulike former for testing av det som utvikles. Noen tester gjøres som nevnt fortløpende av utviklingsteamene. I tillegg kan testing automatiseres med ulike testverktøy som underveis tester koden som produseres. Dette kalles for enhetstesting. Videre testing er såkalte *systemtester* som kan være en kombinasjon av automatiske tester og manuelle tester. Systemtestene gjennomføres som oftest med simulerte testdata. Disse testene skal sikre at alle komponenter fungerer sammen, og både funksjonelle og ikke-funksjonelle krav skal testes. Til slutt gjør kunden *akseptansetesting*. Denne testen gjennomføres som systemtestene men *av* kunden og med kundens testdata. Målet med valideringen er todelt. For det første skal testingen avdekke både syntaksfeil og logiske feil. For det andre skal testene sikre at det som lages er i henhold til de spesifiserte kravene.

Evolusjon kan knyttes til både endringsbehovet underveis i utviklingsprosjektet og til vedlikehold og videreutvikling av programvaren etter at prosjektet er avsluttet. Formålet med aktiviteten er fleksibilitet, slik at programvaren blir best mulig tilpasset brukernes og organisasjonens behov. Underveis i prosjektet er denne fleksibiliteten viktig, fordi kunden ofte ikke vet hva han vil ha før han ser det.

Endringer av noe som allerede er utviklet kan føre til økte kostnader og forsinkelser. Dette gjelder spesielt ved bruk av plandrevne utviklingsmetoder hvor kundens akseptansetesting ikke gjøres før endelig leveranse. Endringshåndteringen er derfor noe som ofte er nøye regulert i de tradisjonelle kontraktene. En måte å redusere endringsbehovet i allerede utviklet programvare er å benytte såkalt *prototyping* underveis i utviklingen. Ved prototyping lages det skjermbilder og simulert funksjonalitet slik at kunden kan gi innspill på prototypene.

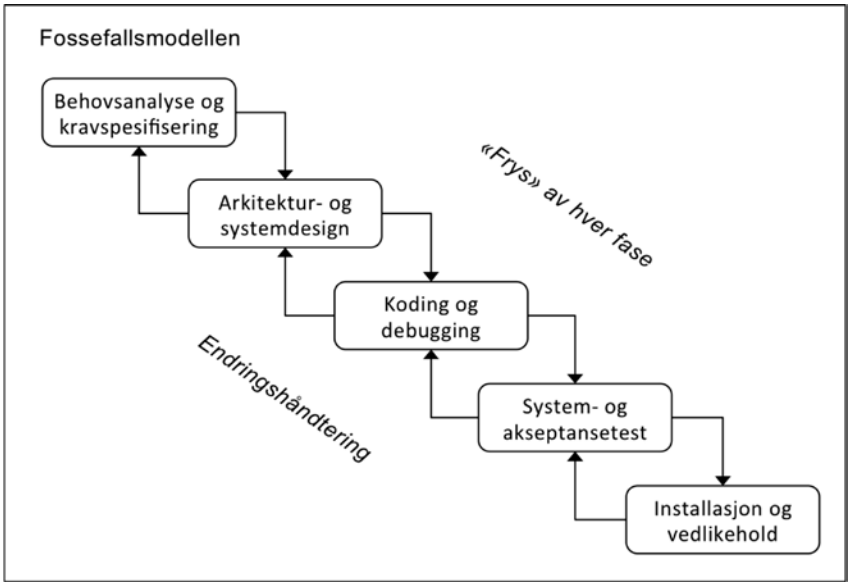
2.3 Plandreven programvareutvikling

Når et resultat skal oppnås ved å gjennomføre et prosjekt, er det ulike aktiviteter som må gjennomføres. I for eksempel entreprise starter det med behovsavklaring i den såkalte *programfasen*. Videre følger *prosjekteringsfasen, byggefasen* og til slutt overlevering og overtakelse. For plandreven programvareutvikling kan aktivitetene være delt opp i tilsvarende faser: behovsavklaring og spesifikasjon, systemdesign, programmering, system- og akseptansetesting og til slutt overlevering og installasjon. De ulike fasene i et prosjekt er delt opp med tanke på at aktivitetene i hver fase henger sammen og at sluttproduktet for hver fase danner underlaget for å starte på neste fase. Sluttproduktet dokumenteres i betydelig grad. En slik oppdeling i faser legger også til rette for prosjektstyring med tanke på planlegging, kontroll og oppfølging av gjennomføringen. Det finnes imidlertid flere måter å organisere fasene på, og for å beskrive dette brukes det ulike modeller.

Det tradisjonelle utgangspunktet er såkalte *plandrevne modeller*. Det vil si at fasene følger hverandre sekvensielt. Innen for eksempel entreprise vil det innebære at byggefasen først starter når prosjekteringsfasen er gjennomført. De ulike plandrevne modellene er varianter av at fasene følger hverandre sekvensielt. En variant er såkalt *fast tracking* som innebærer er at man starter på en fase før den forrige er avsluttet.[86] I totalentrepriser er det ikke uvanlig å starte byggingen parallelt med prosjekteringen for å øke gjennomføringshastigheten.[87] Når det gjelder programvareutvikling er det den såkalte *fossefallsmodellen* (eller også *vannfallsmodellen*) som benyttes hvis prosjektet gjennomføres med en plandreven metode. Også i fossefallsmodellen gjennomføres fasene sekvensielt, men med noe vekselvirkning fra en fase til den forrige slik at det kan gjøres justeringer i for eksempel designdokumenter. Dette skjer typisk via rutiner for endringshåndtering. Utgangspunktet er likevel at resultatet fra hver fase *fryses* slik at nødvendig fremdrift kan sikres innenfor avtalt omfang, tidsplan og kostnad. Derfor illustreres modellen som i figur 2.1, slik at vanskelighetene med å gå tilbake «oppstrøms» tydeliggjøres.

86 Se PMI (2013) s. 43

87 Jf. Barbo (1990) s. 19



Figur 2.1: Et eksempel på en fossefallsmodell for programvareutvikling

2.3.1 Utfordringene med *fossefallsmodellen*

Som nevnt er de plandrevne metodene for prosjektgjennomføring basert på at hver fase avsluttes før neste fase settes i gang. For programvareutvikling med fossefallsmodellen innebærer dette blant annet at en uttømmende kravspesifikasjon må være ferdig før programmeringen starter, og at programmeringen gjøres ferdig før systemtest og kundens akseptansetesting. Disse forutsetningen skaper noen utfordringer.

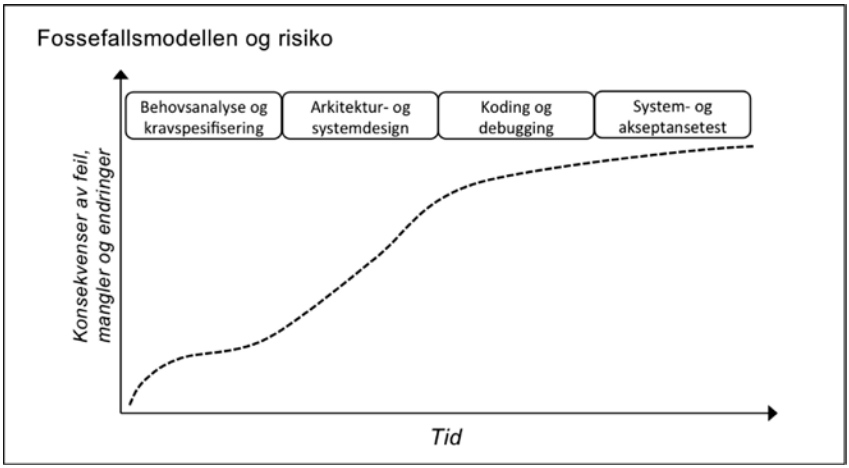
For noen typer systemer kan det være hensiktsmessig, og faktisk mulig, å utarbeide en uttømmende kravspesifikasjon. Dette kan typisk være styringssystemer som har klare krav til funksjonalitet samt liten eller ingen interaksjon med brukere. Det er jo i utgangspunktet alltid ønskelig med stabile og fastlåste krav før programmeringen starter. Dette gjør det enklere å gjennomføre prosjektet og levere et system som oppfyller alle krav. For å sitere McConnell så er «stable requirements […] the holy grail of software development.»[88] Særtrekkene ved programvareutvikling gjør det svært vanskelig å utarbeide en stabil kravspesifikasjon i en tidlig fase av prosjektet. Med en uttømmende kravspesifikasjon blir ofte konsekvensen at det må gjøres mange endringer underveis i

88 Se McConnell (2004) s. 39

prosjektet. Kunden risikerer å få levert et system som ikke tilfredsstiller behovene til tross for at de spesifiserte kravene er oppfylt. I tillegg blir det vanskelig, og kanskje uønsket, fra leverandørens side å initiere endringer. Dette gjør at man mister muligheten til å skape bedre resultater hvis leverandøren finner andre og bedre måter å løse utfordringene på.[89]

Utfordringene med å utarbeide en stabil og uttømmende kravspesifikasjon fører også med seg stor usikkerhet når det gjelder tids- og kostnadsestimering. Jo større usikkerhet det er knyttet til kravspesifikasjonen, jo mindre realistiske er estimatene. I tillegg vil flyten av informasjon mellom for eksempel spesifikasjonsfasen til designfasen og videre til kodingen følge dokumentasjonen som utarbeides. Denne dokumentasjonen vil ikke være dekkende for all informasjon og kunnskap som er opparbeidet i en fase. Dette medfører at *taus kunnskap*[90] fra en fase til den neste ikke overføres.

En annen viktig utfordring er konsekvensene av at system- og akseptansetestingen først gjennomføres etter at programmeringen er ferdig. Feil og mangler som avdekkes i disse testene kan bli kostbare å rette. I tillegg vet, som nevnt, ofte ikke kunden hva han vil ha før han ser det. Kunden vil ofte ønske endringer i funksjonalitet basert på akseptansetestingen. Endringer så sent i prosjektløpet kan også bli kostbare. Figur 2.2 illustrerer sammenhengen mellom fasene og konsekvenser av feil, mangler og endringer i prosjektgjennomføringen.



Figur 2.2: Konsekvensene av feil, mangler og endringer som må rettes, og tidspunktet i prosjektgjennomføringen. (Hentet fra Larman (2004) s. 58)

89 Jf. Henschel (2012) s. 241-242

90 *Taus kunnskap* er kunnskap som ikke eksplisitt er kommet til uttrykk.

Til tross for ulempene ved bruk av fossefallsmodellen, blir den fortsatt mye brukt. Også de tradisjonelle kontraktene for programvareutvikling er tilpasset denne modellen, se nærmere om dette i kapittel 2.5. Fossefallsmodellen ble først beskrevet av Winston W. Royce i 1970.[91] Intensjonen til Royce var ikke å lage en modell som skulle være så strikt som den i praksis er blitt. Modellen, i sin strikse form, fikk likevel stor utbredelse. Spesielt i offentlig sektor (det amerikanske forsvarsdepartementet la grunnlaget) ble fossefallsmodellen populær på grunn av begrensninger i anskaffelsesregelverk og ønsker om kontrakter med fast pris, definert omgang og fast tidsplan. Samtidig med at fossefallsmodellen fikk sin utbredelse ble det også utviklet metoder og modeller for programvareutvikling som ikke kan karakteriseres som plandrevne. Dette er de såkalte *iterative* og *inkrementelle* metoder, som de smidige utviklingsmetodene er basert på.

2.4 Smidig programvareutvikling

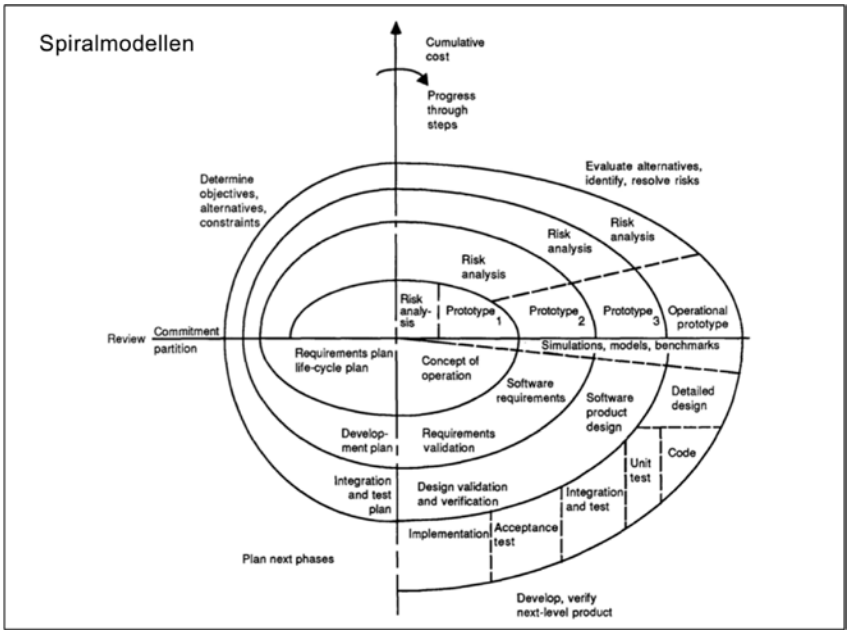
Utgangspunktet for *iterativ* og *inkrementell* utvikling er at de ulike fasene som spesifikasjon, design, programmering og testing gjennomføres for å utvikle kun mindre deler av systemet. Deretter gjentas prosessen for hver ny del av systemet. Dette gjør at kompleksiteten brytes ned i mindre deler, og feil og mangler kan avdekkes tidligere i prosjektet enn ved plandrevne modeller. Når disse metodene også brukes til å få tilbakemeldinger fra brukerne etter hvert som delene utvikles, kalles metodene også *evolusjonære*. Dette var noe Tom Gilb introduserte allerede i 1976.[92] Disse metodene, som kalles *iterative* og/eller *inkrementelle* og/eller *evolusjonære*, kaller jeg i den videre fremstillingen kun *iterative*.

Den første formaliserte *iterative* modellen som ble publisert var Barry Boehms *spiralmodell*.[93] Modellen har fire hovedfaser: spesifikasjon og målsetting, risikovurdering og prototyping, design og koding og planlegging av neste iterasjon. Figur 2.3 viser spiralmodellen. Det som skilte denne modellen fra andre metoder, var fokuset på risikovurdering og risikoredusering.

91 Se nærmere Royce (1970)

92 Se Larman (2003) s. 50

93 Den ble første gang publisert i 1985 og er senere blant annet publisert i Boehm (1988)



Figur 2.3: Barry Boehms spiralmodell, Boehm (1988) s. 64

Til tross for at det på 1970- og 1980-tallet ble gjennomført prosjekter med iterative metoder, var det vanlig å legge til grunn en omfattende kravspesifikasjon som ble utarbeidet før første iterasjon.[94] Derfor var tendensen på 1990-tallet metoder som la mindre vekt på spesifikasjon og enda større vekt på tilbakemeldinger og evolusjon.[95] Sutherland og Schwaber videreutviklet den iterative metoden *Scrum,*[96] som ble benyttet i japansk industriproduksjon på 1980-tallet. Av andre iterative metoder som er kommet til nevner jeg: *Dynamic Systems Development Method* (DSDM), *Rational Unified Process* (RUP), *Extreme Programming* (XP), *Feature-Driven Development* (FDD) og *Lean Software Development*. Undersøkelser av prosjekter som benytter iterative metoder viser at disse resulterer i flere vellykkede prosjekter enn de som baserer seg på fossefallsmodellen.[97] I til-

94 Jf. Larman (2003) s. 53

95 l.c.

96 *Scrum* er et begrep hentet fra ballsporten rugby, og er en måte å stille opp spillerne i en såkalt *klynge.* Ballen droppes i klyngen, og lagene spiller ballen frem og tilbake innen klyngen for å komme seg fremover mot målet, se snl.no/rugby.

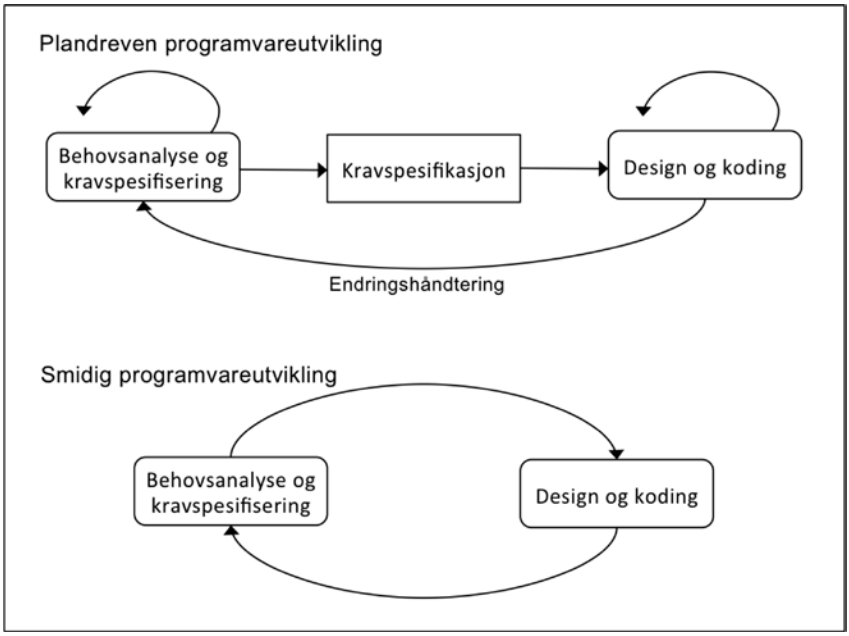
97 Se Larman (2003) s. 54 og Moløkken-Østvold (2005) s. 765

legg viser undersøkelser at tettere samarbeid mellom kunde og leverandør reduserer risikoen for overskridelser med tanke på tid og kostnader.[98]

I 2001 ble begrepet *smidig programvareutvikling*, som på engelsk betegnes som «Agile Software Development», etablert av 17 eksperter på ulike iterative metoder. De opprettet sammenslutningen *Agile Alliance* (www.agilealliance.org) og det såkalte *Manifestet for smidig programvareutvikling*, som på engelsk kalles «Manifesto for Agile Software Development».[99] I tillegg til manifestet ble det laget 12 prinsipper som utdyper den smidig filosofien.[100] *Smidig programvareutvikling* er en fellesbetegnelse på flere ulike metoder som bygger på det smidige manifestet og prinsippene. En av de grunnleggende forskjellene mellom en smidig metode og en plandreven metode er forholdet mellom fasene. I en smidig metode er det en vekselvirkning mellom spesifikasjon, design og koding innen hver iterasjon. I en plandreven metode utarbeides det en uttømmende spesifikasjon som må foreligge før arbeidet med design og koding starter. Eventuelle behov for endringer må da typisk håndteres som en endringsordre. Figur 2.4 illustrerer denne grunnleggende forskjellen.

Undersøkelser viser at de mest brukte utviklingsmetodene i dag er basert på rammeverket som kalles *Scrum*.[101]

98 Jf. Moløkken-Østvold (2007) s. 80
99 Se tabell 1.1 og Beck (2001a)
100 Se vedlegg 5.1 og Beck (2001b)
101 Jf. VersionOne (2014a)



Figur 2.4: Grunnleggende forskjell mellom en plandreven metode og en smidig metode. Min oversettelse av Sommerville (2011) s. 63

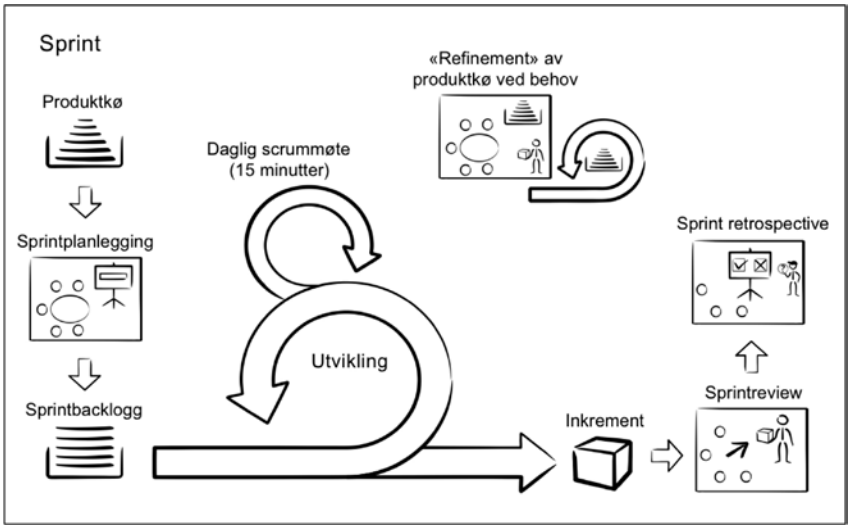
2.4.1 Nærmere om *Scrum*

Som nevnt er *Scrum* basert på en iterativ metode, og er et «[…] *framework* for developing and sustaining complex products [min utheving]».[102] Scrum er et rammeverk fordi den legger større vekt på hvordan den smidige utviklingen kan styres og organiseres, og mindre vekt på teknikker som for eksempel Extreme Programming gjør.[103] Ofte er utviklingsmetodene som benyttes av leverandørene tilpasset organisasjonen og hvilke type løsning som skal leveres, men innenfor rammeverket Scrum. Det at Scrum kan tilpasses ulike teknikker og metoder, og samtidig fungere som et verktøy for styring og organisering, er trolig en av årsakene til at rammeverket har fått stor utbredelse.

Som andre iterative og smidige metoder er resultatet av hver iterasjon i Scrum et *inkrement*. Et inkrement skal tilfredsstille definerte kriterier som angir om inkrementet anses som *ferdig*. På engelsk betegnes inkrementet i Scrum som et «potentially shippable product». Det vil si en ferdig og fungerende del av den

102 Se Sutherland (2013) s. 3
103 Jf. Sommerville (2011) s. 72

totale løsningen. Begrepet *tidsboksing*, som på engelsk kalles «time-boxing», er sentralt i smidige utviklingsmetoder. Tidsboksing er faste tidsrammer for ulike aktiviteter. Hver iterasjon skal ha faste tidsrammer som er lik for alle iterasjonene. Det samme gjelder ulike møter. For møtene er disse rammene maksimumsverdier, men for en iterasjon er tidsboksen fast, for eksempel to eller fire uker. Tidsboksing sikrer fremdrift og kontroll, samt at læringen underveis gjør at man relativt presis kan estimere hvor mye som kan utvikles i hvert sprint. Figur 2.5 illustrerer en iterasjon i Scrum som kalles en *sprint*.



Figur 2.5: En iterasjon med smidig metode basert på rammeverket Scrum

Før jeg beskriver gjennomføringen av en *sprint* sier jeg litt om hvilke artefakter og roller som inngår i Scrum. Min beskrivelse tar utgangspunkt i Sutherlands og Schwabers *Scrumguide*.[104]

Artefakter i Scrum er et *inkrement*, *produktkø* og *sprintbacklogg*. *Produktkøen* er det jeg vil kalle en «dynamisk kravspesifikasjon» fordi innhold og detaljeringsgrad endrer seg fortløpende uavhengig av iterasjonene. Arbeidet med produktkøen starter før første iterasjon og tar utgangspunkt i ønskede målsettinger med løsningen som skal utvikles. Funksjonene, kravene og endringene som ligger i produktkøen skal beskrives, estimeres og prioriteres. Prioriteringen vil som regel ta utgangspunkt i funksjonens eller endringens forretningsverdi for kunden. De elementene i produktkøen som er prioritert høyest må også ha høyest detaljeringsgrad. Dette er fordi det er disse elementene som er gjenstand for

104 Se nærmere Sutherland (2013)

utvikling i neste iterasjon. Hver iterasjon starter med utarbeidelse av en såkalt *sprintbacklogg*. Dette er en delmengde av de høyest prioriterte elementene i produktkøen, samt en plan og en detaljering av hvordan elementene skal utvikles. Omfanget av elementer tilpasses lengden på en iterasjon. Innholdet sprintbackloggen ligger fast, men kan justeres hvis uforutsette ting dukker opp eller at læringen underveis tilsier endringer i for eksempel beskrivelse og estimat. Endringsbehovet i sprintbackloggen vil reduseres etter at flere iterasjoner er gjennomført, fordi utviklerne har mer kunnskap om utviklingshastighet og løsningen som utvikles. Sprintbackloggen kan oppdateres med hvilke elementer som er utviklet.

Rollene som er knyttet til en iterasjon er alle en del av et *scrumteam*, og rollene er *produkteier*, *utviklingsteam* og *scrummaster*. *Produkteieren* er én person og typisk en representant for kunden. Produkteieren er ansvarlig for administrasjon av produktkøen, og målsettingen er å maksimere verdien av produktet og utviklingsarbeidet. Produkteieren kan la andre administrere produktkøen, for eksempel utviklerne, men produkteieren er likevel ansvarlig. Administrasjon av produktkøen består av følgende arbeid:

- Elementene i produktkøen skal prioriteres og uttrykkes klart og forståelig.
- Prioriteringen skal bidra til best mulig oppnåelse av målsettingene.
- Produktkøen skal være åpen og tilgjengelig slik at alle får kunnskap om innhold og hva som skal utvikles.
- Elementene skal inneholde tilstrekkelig informasjon til at utviklerne vet hva som skal lages.

Det er et viktig poeng at produkteieren er én person som er ansvarlig, selv om kundens behov og ønsker utredes og drøftes av flere personer på kundesiden. Det er produkteieren som står for beslutningene overfor scrumteamet. Dette skal blant annet hindre såkalt *scope creep*. Det betyr at utviklingsteamet får endringsønsker fra andre enn produkteiere, og det blir uklart hva som faktisk skal utvikles. Scope creep er en kjent årsak til mangel på kontroll i utviklingsprosjekter.

Utviklingsteamet bør bestå av tre til ni medlemmer som alle er ansvarlige for å utvikle og levere et inkrement etter hver iterasjon. Kompetansen i utviklingsteamet må tilpasses til oppgavene som skal løses. Et viktig aspekt når det gjelder utviklingsteam ved bruk av smidige metoder, er at de er *selvorganiserte*. Det betyr at utviklingsteamet selv er ansvarlig for *hvordan* de utvikler et inkrement i henhold til elementene i produktkøen. Dette innebærer også at utviklingsteamet er ansvarlige for å bidra til at inkrementet oppfyller målsettinger og realiserer verdi for kunden. Utviklingsteamet skal i tillegg estimere og bidra i produkteiers arbeid med produktkøen.

Scrummasteren leder utviklingsteamet og hjelper både medlemmene og andre interessenter til å forstå, lære og gjennomføre programvareutvikling med Scrum. Scrummasteren legger også til rette for den praktiske gjennomføringen, som for eksempel å sørge for at obligatoriske møter gjennomføres som planlagt. I tillegg hjelper hun utviklingsteamet med selvorganisering, progresjon og problemløsning, samt bistår og veileder produkteier med administrasjon av produktkøen.

Gjennomføringen av en *sprint* inkluderer følgende aktiviteter: *sprintplanlegging*, *utviklingsarbeid*, *daglig scrummøte*, *sprintreview*, *sprint retrospective* og *refinement* av produktkøen. Som nevnt skal det konkrete resultatet etter en sprint være et *inkrement*. Hva som skal utvikles i en sprint avklares ved *sprintplanlegging*. Dette er et møte for hele scrumteamet og er tidsbegrenset til maksimum åtte timer. Resultatet etter sprintplanleggingen er sprintbackloggen. Den skal som nevnt inneholde prioriterte elementer fra produktkøen og en detaljering av disse med tilhørende plan for gjennomføring. I tillegg utarbeides det et såkalt *sprintmål*, en beskrivelse som skal være styrende for utviklingsjobben og gi utviklingsteamet veiledning om hensikten med inkrementet. Det er derfor viktig at elementene i sprintbackloggen har en viss sammenheng slik at hele utviklingsteamet kan jobbe mot det samme sprintmålet. Det er utviklingsteamets kapasitet som setter rammene for omfanget av produktkøelementer som kan tas inn i en sprint..

Utviklingsteamet holder *daglige scrummøter* hvor hensikten er å sikre intern kommunikasjon og åpenhet. I tillegg er møtene viktige for å ha fortløpende oversikt over progresjon og mulighet for raskt å avdekke eventuelle hindringer. Møtet er tidsbegrenset til 15 minutter og holdes til faste tidspunkt. Alle utviklerne gir en kort oppdatering om hva som er gjort siden forrige møte, hva som planen før neste møte og om det er noen hindringer som står i veien for å nå sprintmålet.

Sprintreview er et firetimers møte som holdes på slutten av hver sprint. På møtet deltar scrumteamet og andre interessenter som produkteier inviterer. Hensikten er presentasjon og inspeksjon av inkrementet som er laget i sprinten. I tillegg justeres og tilpasses produktkøen i fellesskap for å vurdere prioritering før neste sprintplanlegging. I møtet diskuterer deltakerne hvilke elementer som ble *ferdige*, hva som ikke ble ferdig, hva som fungerte bra, hvilke problemer som oppstod og hvordan disse ble løst.

Sprint retrospective holdes etter sprintreview som en avslutning på en sprint. Møtet har en varighet på tre timer og det er scrumteamet som deltar. Sprint retrospective er et tilbakeblikk på sprinten som er i ferd med å avsluttes. Formålet med tilbakeblikket er at scrumteamet kan evaluere sin egen prestasjon med tanke på medlemmene, samarbeid og prosessen. Basert på evalueringen lages det en plan med forbedringstiltak slik at scrumteamet kan øke effektiviteten.

«Refinement» av produktkøen kalles også *grooming* av produktkøen. Dette er fortløpende arbeid med evaluering, bearbeidelse, detaljering og estimering av elementene i produktkøen. Produkteier samarbeider med utviklingsteamet, og ofte i egne møter. Utviklingsteamet bør ikke bruke med enn ti prosent av sin tid til dette arbeidet.

Til slutt nevner jeg behovet for *skalering* ved større prosjekter. Slik Scrum er beskrevet her er det åpenbart at bare ett utviklingsteam med maksimum ni medlemmer ikke er tilstrekkelig for å utvikle større systemer. Prosjektene blir da organisert med flere utviklingsteam med hver sin scrummaster. Produkteieren håndterer da en eller flere produktkøer. Det kan også benyttes flere produkteiere og med en produkteier som har det overordnede ansvaret.

2.4.1.1 Brukerhistorier

Scrum sier ikke noe om hvordan elementene i produktkøen eller sprintbacklogen skal spesifiseres. En måte å spesifisere elementene er ved hjelp av såkalte *brukerhistorier*, som på engelsk kalles «user stories». Konseptet med brukerhistorier ble først utviklet av Kent Beck, som en del av den smidige utviklingsmetoden Extreme Programming,[105] og er videreutviklet av blant annet Mike Cohn.[106] Brukerhistorier er nå vanlig i prosjekter som benytter smidige utviklingsmetoder, og er blitt en de facto standard i Scrum.

Brukerhistorier er en måte å spesifisere hva de ulike brukerne av systemet forventer og ønsker av systemet som skal utvikles. En vanlig tilnærmingsmåte er først å definere hvilke brukergrupper som skal bruke systemet på en eller annen måte. For hver brukergruppe defineres hvilke forventninger, ønsker og mål denne gruppen har til bruken av systemet. Dette omsettes deretter til overordnede brukerhistorier som kalles *epos*, som på engelsk kalles «epic». Et epos kan uttrykkes som følger: *Som administrator ønsker jeg å administrere brukere, slik at jeg har kontroll på hvem som har tilgang til systemet.* Denne beskrivelsen må splittes opp i flere brukerhistorier med høyere presisjon og detaljeringsgrad. En brukerhistorie kan beskrives som følger: *Som administrator ønsker jeg å opprette en ny bruker, slik at jeg enkelt kan gi nye brukere tilgang.* Vi ser at brukerhistorien gir mer presis informasjon om hva brukeren ønsker av systemet. En slik brukerhistorie kan legges inn som et element i produktkøen. En brukerhistorie vil typisk bli omgjort til tekniske oppgaver ved sprintplanleggingen og legges inn som elementer i sprintbacklogen. Flere brukerhistorier som logisk hører sammen, kan grupperes til et *tema*.

Hensikten med brukerhistorier er blant annet å ha en mer uformell beskrivelse av hvilke behov systemet skal dekke, og få tydelig fram *hensikten* med det som skal utvikles. Dette hjelper scrumteamet til å holde fokus på at program-

105 Se blant annet Beck (2004)
106 Se Cohn (2004)

varen som utvikles har verdi for kunden og brukerne. Dessuten blir det lettere for utviklerteamet å sette seg i brukernes situasjon. Brukerhistoriene har som regel et fast format på behovene som beskrives:

Som en <bruker>, ønsker jeg <hva>, slik at <hvorfor>.

I vedlegg 5.3 finnes det eksempler på både epos og brukerhistorier. Det er også vanlig å knytte visse akseptansekriterier til hver brukerhistorie. Akseptansekriteriene brukes ved testing, i tillegg til at de øker brukerhistoriens detaljeringsgrad. Akseptansekriterier for eksempelet nevnt over kan være:

- krav til minimumsinformasjon om ny bruker
- det skal ikke gå an å legge inn ny bruker uten minimumsinformasjon
- samme bruker kan ikke legges inn flere ganger

Akseptansekriteriene utvikles underveis i arbeidet med produktkøen, og vil som regel være et av kravene til at et inkrement skal bli definert som *ferdig.*

2.4.1.2 Estimering

Den grunnleggende usikkerheten knyttet til programvareutvikling gjør estimeringsarbeidet vanskelig. For plandreven programvareutvikling og kontrakter med fast pris og fast tid er det helt nødvendig å estimere det som skal lages, men usikkerheten knyttet til estimatene er store. Se nærmere om usikkerhet i underkapittel 2.5.3. Selv om et prosjekt benytter smidige utviklingsmetoder, er estimering nødvendig. Det er nødvendig for å kunne planlegge delleveransene, og kunne si noe om hvor stort prosjektet er når det gjelder tid og kostnad. Det finnes mange estimeringsteknikker, og en del forskning på hvilke teknikker som egner seg for de smidige metodene. En vanlig teknikk er såkalt *planning poker*. Planning poker er gruppeestimering av typisk epos og brukerhistorier. Estimeringen kan gjøres underveis i arbeidet med produktkøen, og i planleggingen av en sprint. I planning poker brukes som regel en relativ måleenhet som kalles *storypoints*, og estimeringstallet på en brukerhistorier kan typisk hentes fra følgende rekke: 1, 2, 3, 5, 8, 13, 20, 40 og 100. Hvert medlem av estimeringsgruppen, som typisk vil være scrumteamet, spiller ut et kort med sitt estimat på en av brukerhistoriene. Som regel har ikke alle medlemmene den samme oppfatningen av kompleksiteten og omfanget til brukehistorien, og de spiller ut ulike tall. Dette resulterer i diskusjon og drøftelse rundt brukerhistorien, og det spilles deretter på nytt for samme brukerhistorie. Denne iterative estimeringen fører til mer korrekte estimeringstall og viktige diskusjoner rundt utviklingen av en brukerhistorie. I tillegg er dette en estimering «top-down», hvor man starter med epos og deretter brukerhistorier. Ved plandreven utvikling estimeres det

som regel motsatt, slik at alle definerte utviklingsoppgaver estimeres, og deretter aggregeres.

2.4.2 Oppsummering

I tabell 2.1 oppsummerer jeg ulikhetene mellom plandrevne og smidige metoder. En viktig forskjell som jeg vil fremheve, er de smidige metodenes grunnleggende målsetting om å styre mot økt verdi for kunden gjennom hele prosjektet. I en plandreven metode er det kundens ansvar å sørge for at verdien som skal realiseres med utviklingsarbeidet, blir ivaretatt gjennom en detaljert og uttømmende kravspesifikasjon. Leverandøren har ansvaret for å levere systemet i henhold til kravspesifikasjonen. Smidige metoder tar utgangspunkt i de verdiene og målsettingene kunden har ved spesifikasjonen i oppstarten av hver iterasjon. Inkrementet som leveres etter hver iterasjon evalueres. Kunden har derfor en vesentlig delaktighet i hele utviklingsperioden for å bidra med tilbakemeldinger og innspill på hvordan systemet skal utvikles slik at ønsket verdiøkning kan realiseres.

Tabell 2.1: Sammenlikning av plandreven og smidig utviklingsmetode

Plandreven utviklingsmetode	**Smidig utviklingsmetode**
Sekvensielle faser, hvor en fase må avsluttes før den neste tar til.	Alle fasene gjennomføres flere ganger – iterativt.
Omfattende kravspesifikasjon og design utarbeides og er uttømmende før kodingen starter.	Kravspesifikasjon og design utføres innen hver iterasjon i vekselvirkning med kodingen.
Systemet er ferdig når siste fase er avsluttet og alle kravene er oppfylt.	Systemet ”gror” gradvis med inkrementer som skaper verdi for kunden.
Endringer i kravspesifikasjon og design gjøres med formaliserte endringsprosedyrer.	Endringer i kravspesifikasjon og design skjer i vesentlig mindre grad siden disse utarbeides i samme iterasjon i en vekselvirkning med kodingen.
Endringer er uønsket og kan bli kostbare.	Endringer er ønsket slik at endrede eller nye muligheter kan gi økt verdi for kunden.
Prosjektets vellykkethet måles på graden systemet oppfyller kravspesifikasjonen, budsjett og tidsplan.	Vellykkethet måles fortløpende på levert verdi for kunden.
Kundens rolle er viktig – spesielt under utarbeidelsen av kravspesifikasjon og akseptansetesting.	Kundens medvirkning er kritisk, og fordrer samarbeid under hele prosjektperioden.
Organiseringen er byråkratisk med høy grad av formalisme og formell kommunikasjon.	Fleksibel og deltakende organisering med høy grad av samarbeid og uformell kommunikasjon med selvorganiserte utviklingsteam.
Prosjektstyringen er knyttet til prosessene.	Prosjektstyring er knyttet til personer og selvorganiserte team.
Betydelig dokumentasjon underveis, og mangel på taus kunnskap.	Lite dokumentasjon underveis, men stor grad av taus kunnskap.

2.5 Kontraktsforhold

I dette kapittelet drøfter jeg noen sider ved den tradisjonelle kontraktsutformingen i lys av programvareutvikling generelt. Med tradisjonell kontraktsutforming sikter jeg til tilvirkningskontrakter hvor leverandøren påtar seg en resultatforpliktelse, og ikke til kontrakter for innleie av konsulenttjenester, se figur 1.1.

2.5.1 Tradisjonelle kontrakter

Problemstillingen i denne oppgaven er knyttet til spesifikasjon av kontraktsgjenstanden og gjelder dermed den primære typen av kontraktsbestemmelser.[107] Det vil si bestemmelser som beskriver partenes ytelsesplikter – både realforpliktelser og pengeforpliktelser. Drøftelsene tar utgangspunkt i leverandørens realforpliktelse. Som nevnt i avgrensningen handler programvareutvikling om tilvirkning av programvare, og kontraktene kategoriseres som tilvirkningskontrakter. De vanlige standardkontraktene som benyttes for tradisjonelle programvareutviklingsprosjekter er ensidig utarbeidede vilkår fra Difi, IKT-Norge og DND, se avsnitt 1.5.1.1. DNDs PS2000 standardkontrakter er basert på iterative utviklingsmetoder, som jeg går nærmere inn på i avsnitt 2.5.1.1. Standardkontraktene er basert på bakgrunnsretten, men har også hentet inspirasjon fra de norske fabrikasjons- og entreprisekontraktene.

Både SSA-U og IKT 2010 bygger på faser i en fossefallsmodell kombinert med et fast avtalt vederlag (fast pris). SSA-U definerer tre faser: spesifiseringsfasen, utviklingsfasen og leveringsfasen, jf. pkt. 1.1. Den forutsetter at utviklingsfasen ikke starter før kunden har godkjent detaljspesifiseringen som leverandøren skal utarbeide i spesifiseringsfasen, jf. pkt. 2.2.1. I pkt. 2.2.2 kalles denne «*endelig* kravspesifikasjon [min utheving]». Denne formuleringen tydeliggjør at kravspesifikasjonen er en uttømmende beskrivelse av hva som skal lages, og at den skal være fast under hele utviklingsfasen. Det er kundens kravspesifikasjon og leverandørens løsningsspesifikasjon (begge brukes i den offentlige anskaffelsesprosessen) som skal ligge til grunn for detaljspesifiseringen. En fast og uttømmende kravspesifikasjon er også forutsatt i IKT 2010. Ifølge avtalen skal kunden utarbeide funksjonskrav før avtalen inngås, jf. pkt. 3.3.2, og begge parter skal i fellesskap utarbeide en såkalt produktspesifikasjon, jf. 3.1.2. Videre i pkt. 3.3.2 står det at «[a]lle funksjonskrav og spesifikasjoner skal være uttømmende regulert […]». Kravspesifikasjonen er et sentralt dokument hvor det blir forsøkt entydig å definere hva leverandøren forpliktes til å levere. I henhold til SSA-U og IKT 2010 er leverandøren forpliktet til utvikle programvare som dekker alle krav som er spesifisert, og med eventuelle endringer. Dette kommer frem i bestemmelser om leverandørens plikter sammenholdt med vilkår for mislighold.

107 Jf. Haaskjold (2013) s. 94-95

I SSA-U står det i pkt. 5.1 at «[l]everandøren har ansvar for at den samlede leveransen (den helhetlige løsningen) dekker de funksjoner og krav som er spesifisert i avtalen». I pkt. 11.1 regnes det som mislighold hvis leveransen ikke dekker funksjoner og krav som er avtalt. Tilsvarende bestemmelse har IKT 2010 i pkt. 16.2.2, pkt. a om at det foreligger en mangel hvis produktet ikke oppfyller kravene i produktspesifikasjonen. Disse bestemmelsene viser at leverandøren påtar seg en resultatforpliktelse.

Kontraktenes utforming, med tanke på kravspesifikasjonen, kan oppsummeres med Nicolai Dragsteds sitat: «Kravspecifikationen gøres til krumtap i kontrakten, og fokus for de juridiske håndgreb er at sikre etablering af netop det resultat, der er detaljeret beskrevet i kravspecifikationen.»[108] Likevel er det slik at uttømmende og faste kravspesifikasjoner ved prosjektstart sjelden forblir uforandret, se underkapittel 2.5.3.

2.5.1.1 Standardkontraktene *PS2000*

DND forvalter standardkontraktene *PS2000,* som finnes i flere varianter: *PS2000 Standard* (første versjon lansert i 1999), *PS2000 Smidig* (lansert i 2009) og *PS2000 SOL* (Smidige Oppdragsbaserte Leveranser). Den siste ble lansert i september 2013 og skal kunne brukes i prosjekter med smidig programvareutvikling hvor spesifikasjoner og definert omfang ikke utarbeides før utviklingen tar til, se underkapittel 3.3.2.

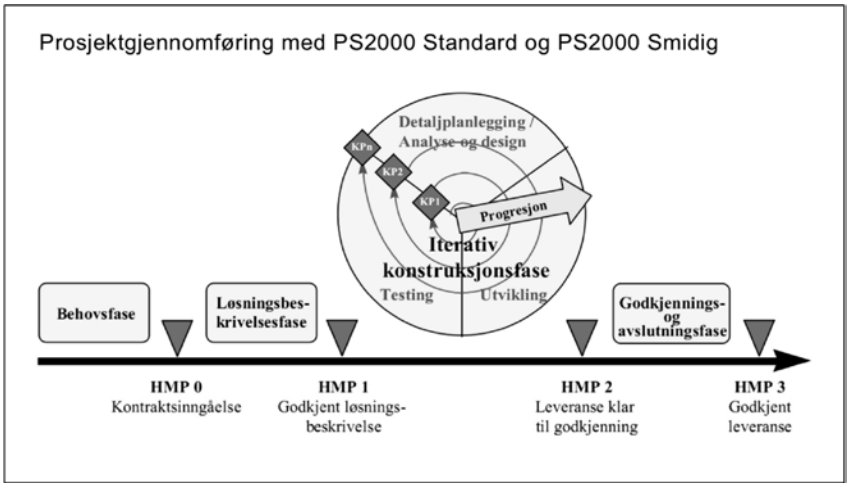
PS2000 Standard og PS2000 Smidig legger begge opp til en iterativ utvikling med stor vekt på risikovurderinger for hver iterasjon – på samme måte som Bohem beskrev risikohåndtering i sin spiralmodell, jf. figur 2.3. Begge standardkontraktene består av tre deler med tilhørende veiledere. Del I (Kontraktsdokument) angir partene, samt gir oversikt over kontraktsdokumentene og strukturen. Del II (Generelle kontraktsbestemmelser) inneholder standardvilkårene, mens del III (Kontraktsbilag) består av bilagene som delvis er utfylt med forslag til innhold. I PS2000 Standard og PS2000 Smidig er del I og II like – forskjellene ligger i bilagsinnholdet og veilederne.

Begge standardkontraktene legger opp til en sekvensiell faseinndeling av gjennomføringen: løsningsbeskrivelsesfasen, konstruksjonsfasen og godkjennings- og avslutningsfasen, jf. del II, pkt. 1.1, andre avsnitt. Det er imidlertid to forhold som skiller kontraktene fra SSA-U og IKT 2010. Det ene er fokuset på at usikkerheten reduseres i konstruksjonsfasen ved at «[…] den trinnvise utviklingen i hovedsak er iterativ […]», jf. del II, pkt. 1.1, tredje avsnitt. Faseinndelingen med iterativ konstruksjonsfase vises i figur 2.6 med kontraktenes egen illustrasjon. Det andre forholdet som skiller kontrakten fra SSA-U og IKT 2010 er at det legges opp til en prismodell med *målpris*.[109] PS2000 Smidig skiller seg fra PS2000

108 Dragsted (2012) s.15

109 Se nærmere om prismodeller i underkapittel 3.6.2

ved at den har en egen detaljert veiledning om hvordan kontrakten kan brukes ved smidig programvareutvikling. Dessuten er utviklingsprosessen med Scrum detaljert regulert i bilagsdelen.



Figur 2.6: Faseinndeling med iterativ konstruksjonsfase med PS2000. (Hentet fra del III, pkt. C.1)

Til tross for at begge kontraktene tar utgangspunkt i en iterativ gjennomføring i konstruksjonsfasen, er det en forutsetning at det utarbeides et uttømmende og fast definert omfang i *løsningsbeskrivelsesfasen* som skal ligge til grunn for konstruksjonsfasen, jf. del II, pkt. 3.3.1, andre avsnitt. Eventuelle ønskede endringer i løsningsbeskrivelsen i konstruksjonsfasen (eller senere) må skje med formaliserte endringsprosedyrer i henhold til pkt. 3.6. Disse forholdene gjør at begge kontraktene må plasseres i samme kategori som SSA-U og IKT 2010 – altså de jeg omtaler som *tradisjonelle kontrakter* som er basert på en plandreven metode. Likevel er dette standardkontrakter som, etter mitt syn, er bedre egnet for programvareutvikling enn SSA-U og IKT 2010. Spesielt siden det er lagt inn såkalte *kontrollpunkter* etter hver iterasjon, jf. del II, pkt. 3.4.5. Disse kontrollpunktene tilsvarer den aktiviteten i Scrum som kalles *sprintreview*, og i PS2000 Smidig del III, pkt. C.4.3.6 er det lagt inn viktige elementer som presentasjon av inkrementet, kundens godkjenning og risikoevaluering. Disse kontrollpunktene legger også til rette for læring og erfaringsutveksling fra forrige iterasjon. Dette muliggjør løpende kontroll, og det avdekkes raskt om prosjektet er i ferd med å ta en uønsket retning. I begge kontraktene har leverandøren et resultatansvar på sam-

me måte som i SSA-U og IKT 2010, men kundens medvirkningsplikter er betydelig utvidet.

2.5.2 Risiko og risikofordeling

Risiko og *risikofordeling* er grunnleggende begreper i kontraktsretten når partenes plikter og rettigheter skal fastlegges. Som regel er det først når partene er kommet i en uønsket situasjon med uenighet om fremdrift, omfang eller kvalitet hvor spørsmålet om risikofordeling blir satt på spissen. Hvordan kontrakten regulerer risikofordeling vil være avgjørende for å konstatere om det foreligger kontraktsbrudd eller ikke, og eventuelt hvilke krav og sanksjoner partene kan bli stilt overfor. Kontraktens regulering av risikofordeling vil også hjelpe partene før det har oppstått en konflikt, fordi det skaper en forutsigbarhet om hvilken risiko partene påtar seg, og må ta hensyn til i prosjektgjennomføringen. Som nevnt vil en smidig utviklingsmetode bygge på tett samarbeid og betydelig innsats fra begge parter, og risikofordelingen vil forrykkes sammenliknet med en plandreven utviklingsmetode. Derfor er det viktig å få frem hva som ligger i begrepene og hvordan tradisjonell risikofordeling er regulert i et tilvirkningsforhold. En slik avklaring er nødvendig for å ha et utgangspunkt når risikofordeling ved smidig utviklingsmetode skal vurderes.

Begrepet *risiko* brukes i mange sammenhenger og har ulike betydning. Knyttet til prosjektgjennomføring definerer Project Management Institute (PMI) risiko som følger: «Project risk is an uncertain event or condition that, if it occurs, has a positive og negative effect on one or more project objectives such as scope, schedule, cost, and quality. A risk may have one or more causes and, if it occurs, it may have one or more impacts.»[110] Risiko i et prosjekt er altså en hendelse med en viss sannsynlighet om den inntreffer eller ikke, og den kan ha flere årsaker og få flere konsekvenser. Definisjonen til PMI får også frem at hendelsen også kan ha positive konsekvenser, men som regel er de negative sidene vi forbinder med risiko. Usikkerheten i et programvareutviklingsprosjekt er, som tidligere påpekt, stor. Det er derfor høy sannsynlighet for at det oppstår uønskede situasjoner, som får konsekvenser for partene. Nå begrepene risiko og risikofordeling skal legges til grunn for kontraktsregulering må det gjøres en nærmere juridisk avklaring av begrepene.

Det sentrale spørsmålet er hvem av partene som skal bære de negative konsekvensene når kontraktsforholdet utvikler seg i uønsket retning. Den type risiko, som ligger til grunn for risikofordelingen i kontrakter jeg drøfter i denne oppgaven, er den såkalte *vederlagsrisikoen*. Utgangspunktet for innholdet i vederlagsrisikoen er at leverandøren har risikoen for at han ikke oppfyller innholdet i kontrakten på en slik måte at han mister retten til vederlag. For kunden innebærer vederlagsrisikoen at, til tross for oppfyllelsessvikt fra leverandørens side,

110 Jf. PMI (2013) s. 310

kunden kan risikere å måtte betale for en ytelse som ikke er i henhold til kontrakten.

En oppfyllelsessvikt fra leverandørens side, hvor årsaken til svikten ikke skyldes noe kunden har risikoen for, er et *kontraktsbrudd*.[111] Kunden har også plikter i et kontraktsforhold, som for eksempel å betale vederlag og yte medvirkning. Svikter kunden i å oppfylle disse pliktene, og dette ikke skyldes forhold leverandøren har risikoen for, er dette også et kontraktsbrudd. Dette omtales ofte som *kreditormora*.

Partenes kontraktsbrudd kan utløse sanksjoner. Dette kan illustreres med kjøpslovens[112] §§ 22 og 30 (kontraktsbrudd av leverandør) og § 51 (kontraktsbrudd av kunden). I enklere kontraktsforhold, som for eksempel kjøp av løsøre, gir kjøpsloven klare regler for risikofordelingen mellom kjøper og selger. Dette gjelder spesielt risikoen for fysisk skade på ting, jf. kjøpslovens kapittel III. Når det gjelder oppfyllelsessvikt som ikke skyldes fysisk skade og kontrakten gjelder et tilvirkningskjøp, blir risikobildet mer uklart. Risikofordelingen må derfor vurderes for de enkelte kontraktstyper.[113]

I komplekse tilvirkningskontrakter som programvareutvikling, entreprise og fabrikasjon kan det være mange hindringer for oppfyllelse etter kontrakten. Hver av partene har flere plikter, og oppfyllelse av kontrakten bærer preg av et samarbeid mellom partene. Dette gjør at vurderingen av vederlagsrisikoen blir mer komplisert.[114] Risikofordelingen i slike kontrakter må ta utgangspunkt i regelen om «[…] at hver av partene har risikoen for svikt på eget funksjonsområde.»[115] Dette utgangspunktet er den såkalte *funksjonsdelingen*, og henger nøye sammen med hvordan prosjektet er organisert og pliktene fordelt mellom partene. I et byggeprosjekt er det ofte slik at byggherren skal stå for prosjekteringen, og entreprenøren skal utføre byggingen i henhold til det som er prosjektert. Hvis for eksempel byggverket har mangler som skyldes svikt i prosjekteringen, er det byggherren som har risikoen for dette. Dette ble allerede slått fast i Rt. 1917 s. 673.[116] I NS 8405 er dette regulert i pkt. 19.2. Hvis byggeprosjektet organiseres som *totalentreprise* er det entreprenøren som også står for prosjekteringen og har dermed risikoen for svikt i denne, jf. NS 8407 pkt. 16.1. Ved programvareutvikling er det vanskelig å trekke et slikt skille mellom prosjektering og utførelse, jf. underkapittel 2.1.5. Leverandøren skal designe og programmere programvare i henhold til kundens spesifikasjon av behov og funksjonalitet. I SSA-U er dette regulert i pkt. 5.1 og pkt. 6.1. Ved en slik funksjonsdeling har leverandøren i stor

111 Se Hagstrøm (2011) s. 328
112 Kjøpsloven (1988)
113 Se Hagstrøm (2011) s. 333
114 Jf. Krokeide (1977) s. 581-582
115 Jf. Hagstrøm (2011) s. 333
116 Rt. 1917 s. 673

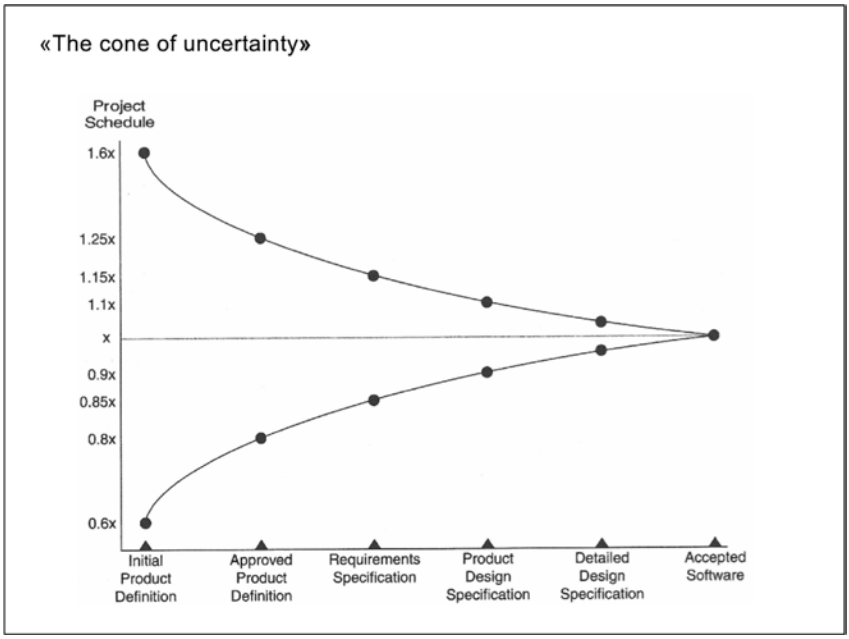
grad risikoen for at resultatet nås, og kontraktsforholdet ligger tett opp til byggebransjens totalentreprise.[117]

2.5.3 De uunngåelige endringene

Særtrekkene ved programvareutvikling, se kapittel 2.1, gjør at uttømmende og faste kravspesifikasjoner ikke forblir uforandret i prosjektperioden. Usikkerheten knyttet til kompleksiteten og at kundens ulike funksjonsbehov oppdages først når systemet blir levert, gjør at kravspesifikasjonen må endres underveis. Jo lenger kontraktsforholdet varer, jo større sannsynlighet er det for at eksterne forhold, som markeds- og lovendringer, krever endringer i kravspesifikasjonen. I tillegg er det som regel generelt ønsker om endringer, fordi det faktisk er mulig å gjøre endringer, selv om utviklingen har kommet et godt stykke på vei.[118]

Den største usikkerheten er imidlertid knyttet til initieringsfasen og oppstarten av prosjektet. Usikkerheten gjør seg gjeldende når en kravspesifikasjon skal utarbeides, og dermed vil også estimat og tidsperspektivet være vanskelig å fastslå. McConnell kaller den varierende graden av usikkerheten i programvareutviklingsprosjekter for «the cone of uncertainty». Modellen ble opprinnelig utviklet av Boehm i 1981. Den viser hvordan usikkerheten, med tanke på tidsaspektet ved utviklingen, avtar jo mer kunnskap prosjektet har om fremdrift og hva som skal utvikles.[119] Figur 2.7 gjengir Boehms modell. Figuren viser at i den initielle fasen kan estimert tid for utviklingen variere fra 160 prosent over estimat, til 60 prosent under estimat. Andre opererer med tilsvarende tall, uavhengig av prosjekttype, i størrelsesorden 50 prosent, og 75/25 prosent.[120]

117 Jf. Føyen (2006) s. 58
118 Se Brooks Jr. (1995) s. 185
119 Se Cohn (2005) s. 3-4
120 Se Karlsen (2013) s. 346 og PMI (2013) s. 201



Figur 2.7: Boehms «cone of uncertainty» hentet fra Cohn (2005) s. 4

Denne usikkerheten har ført til at såkalt *evolusjon* er blitt en viktig aktivitet i programvareutvikling. Evolusjon er et behov for endringer underveis i prosjektgjennomføringen, som beskrevet i kapittel 2.2. Bakgrunnsretten regulerer ikke endringsadgangen, annet enn regler for å «motvirke sterk urimelighet eller ubalanse i kontraktsforholdet […]».[121] Derfor har det fleste standardkontrakter for tilvirkning bestemmelser om endringshåndtering. De norske fabrikasjonskontraktene har vært en inspirasjon for entreprisekontrakter og programvarekontrakter når det gjelder endringshåndtering. Både SSA-U og IKT 2010 har detaljerte regler og prosedyrer for gjennomføring av endringer etter kontraktsinngåelsen. Dette kan gjelde flere typer endringer, men det er sannsynligvis endringer i og tillegg til spesifikasjonene som står for hovedtyngden av endringsordrene. Endringsreglene i disse kontraktene omtaler Kaasen som *dynamisk kontraktsrett,* og legger til rette for at kontrakten fungerer som et verktøy mellom partene til tross for endrede forutsetninger.[122]

Endringer av regulerte forhold i kontrakten etter inngåelse trenger en viss grad av formalisme for å sikre at endringene blir forstått som en del av kontrak-

121 Jf. Kaasen (2005) s. 238
122 Se Kaasen (2005) s. 240

ten. I tillegg vil endringsreglene få frem eventuelle konsekvenser av endringene med tanke på kostnader og fremdrift. Endringssystemet blir også benyttet for saksbehandling av kontraktsbrudd. Ofte er det knyttet preklusive frister til endringsprosedyrene. Slike frister bidrar til å fremtvinge avklaringer slik at fremdriften holdes oppe, og det kan lette prosjektstyringen.[123] For at endringssystemene skal fungere som forutsatt, er det viktig at partene følger prosedyrene slik de er regulert i kontrakten. Hvis endringsreglene blir for omfattende og kompliserte, kan det føre til at partene ikke følger systemet som forutsatt i kontrakten. Det er ikke uvanlig at partene mangler både kompetanse og vilje til å følge de formaliserte endringsreglene. Dette kan, ved en eventuell tvist, føre til at domstolen løser konflikten på en annen måte enn det partene hadde forventet.[124]

Utfordringene med formaliserte endringsprosedyrer øker i takt med omfanget av endringsordrer i et prosjekt. Et eksempel er en dom fra tingretten i det havarerte konsolideringsprosjektet av Telenors serverpark.[125] Retten pekte på at en av utfordringene i prosjektet var det «betydelig antall» endringsordrer – over hundre stykker i en periode på drøyt to år. I tillegg ble det gjort avvik fra endringsprosedyrene. Som et annet eksempel på omfang av endringsordrer var Kommunal- og moderniseringsdepartementets anskaffelse av programvare for administrasjon av valggjennomføring i Norge – inkludert forsøksprosjekt med stemmegivning via internett.[126] Kontraktssummen var på rundt 30 millioner kroner, og behandlet 195 endringsordrer.[127] Det er liten tvil om at oppfølging av formaliserte endringsprosedyrer kan være krevende for partene når omfanget er stort. Derfor må partenes behov være styrende for hvordan reglene utformes i den enkelte kontrakt.[128] For fabrikasjonskontraktene kan det nok være nødvendig med detaljerte prosedyrer og strenge preklusjonsregler. I disse prosjektene er det mye som står på spill når det gjelder kostnader. For eksempel må arbeid som ikke blir ferdig på land, sluttføres til havs. Dette får betydelige kostnadsøkninger.

Når det er så stor grad av usikkerhet ved programvareutviklingen, kan formaliserte endringsprosedyrer bli omfattende. I tillegg kan det være at det trengs nye endringer av tidligere endringsordrer, og den praktiske anvendelsen nærmer seg bristepunktet. En annen utfordring med endringsordrene er at en leverandør som skal levere til fast pris, kan være fristet av å redusere risikoen for tap ved hjelp av endringsordrer. Dette er fordi endringsordrer ofte utløser krav om

123 Jf. Kaasen (2009) s. 169

124 Se Knag (2010) s. 106-107

125 TOSLO-2008-126000 Dommen ble anket, men partene ble forlikt før videre domstolsbehandling.

126 Kommunal- og moderniseringsdepartementet (2011)

127 Se vedlegg 5.2

128 Jf. Kaasen (2005) s. 261

tilleggsbetaling.[129] Tilleggsbetaling for mange endringsordrer gjør også at den forutsigbarheten kunden i utgangspunktet har, med tanke på pris i en fastpriskontrakt, blir redusert. Dragsteds og Klints undersøkelse underbygger at endringshåndtering er et vesentlig problemområde i en it-kontrakt.[130] Utfordringene med endringshåndteringene er spesielt knyttet til plandrevne utviklingsmetoder med uttømmende spesifikasjon av programvaren ved kontraktsinngåelse. De smidige metodene er utviklet blant annet med tanke på å redusere behovet for formaliserte endringer, fordi programvaren i liten grad er spesifisert i kontrakten. Detaljspesifisering skjer kun ved oppstart av hver iterasjon, og endringsbehovet innenfor en iterasjon er lite. Jeg går nærmere inn på behovet for endringsordrer i kontrakter for smidig programvareutvikling i underkapittel 3.6.4.

129 Se Atkinson (2011) s. 3

130 Se Dragsted (2013) s. 30

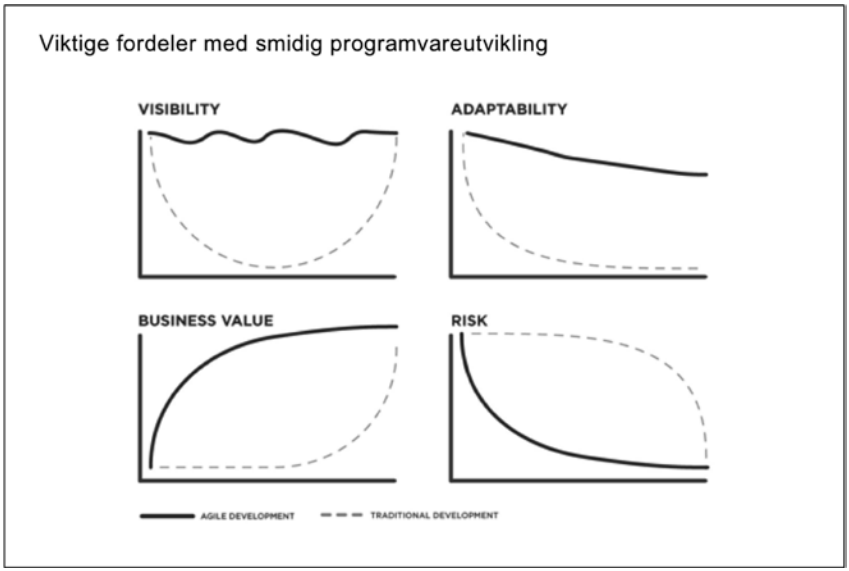
3 Kontraktsregulering av smidig programvareutvikling

3.1 Innledning

Jeg har tidligere i oppgaven sett på ulike aspekter knyttet til programvareutvikling, utviklingsmetoder og tradisjonelle tilvirkningskontrakter. Jeg gikk spesielt inn på smidig utviklingsmetode og beskrev rammeverket *Scrum*. Ulikhetene mellom en plandreven utviklingsmetode og en smidig utviklingsmetode ble oppsummert i tabell 2.1. Kort fortalt kan følgende fordeler oppnås ved å bruke en smidig utviklingsmetode:

- fleksibilitet når det gjelder ønskede og uønskede endringer
- økt forretningsverdi og større muligheter for gevinstrealisering
- økt åpenhet og synlighet som gir løpende kontroll og styring
- bedre læring, samarbeid og kommunikasjon
- en lavere totalrisiko i prosjektet
- økt produktivitet og økt ressurseffektivitet
- høyere kvalitet med lavere kostnader og tidsforbruk

De viktigste fordelene og forskjellene mellom utviklingsmetodene, er illustrert i figur 3.1.



Figur 3.1: Fordeler med smidig programvareutvikling. (VersionOne (2014b)

Som nevnt er én måte å gjennomføre et prosjekt, som benytter smidig programvareutvikling, at kunden selv tar det meste av risikoen og kun leier inn ressurser (konsulenter). En forutsetningen ved en slik løsning er at kunden har tilstrekkelig kompetanse – enten egne ansatte eller innleide konsulenter som kan sikre en vellykket prosjektgjennomføring. Leverandøren av konsulentressursene i slike prosjekt påtar seg en innsatsforpliktelse, men gir ingen forutsigbarhet for kunden når det gjelder resultatet. Imidlertid vil det ofte være slik at kundene ønsker en viss forutsigbarhet både med tanke på resultat og kostnader. Og man sikter seg derfor inn på de tradisjonelle kontraktene med en klar resultatforpliktelse for leverandøren. Spesielt i offentlig sektor, hvor anskaffelsesregelverket er styrende for hvordan anskaffelsesprosessen skal gjennomføres, velges ofte en kontrakt med resultatansvar for leverandøren, for eksempel SSA-U.

Som nevnt i innledningen oppgir 81 prosent av it-bransjen (av de som faktisk benytter formelle utviklingsmetoder) at de benytter smidige utviklingsmetoder.[131] Brevik og Grønli har også undersøkt i hvilken grad de som oppgir at de benytter smidige utviklingsmetoder faktisk gjennomfører prosjektene i henhold til det smidige manifestet med tilhørende prinsipper. De konkluderer med at bransjen kun i begrenset grad følger de smidige prinsippene.[132] De baserer sine

131 Se Brevik (2013) s. 19-20
132 ibid. s. 25

konklusjonene på flere forhold, men det er et område jeg vil trekke frem som spesielt interessant: Over halvparten av respondentene i undersøkelsen oppgir at de må følge en fast kravspesifikasjon ved utviklingen, i motsetning til en dynamisk produktkø som ligger til grunn for sprintbackloggen.[133] Et slikt avvik fra den smidige utviklingsmetoden bryter helt med prinsippene om fleksibilitet, som er en grunnleggende forutsetning i smidigtankegangen. Dette funnet er etter min mening bekymringsfullt med tanke på sjansen for å lykkes med smidig programvareutvikling. Undersøkelsen viser også at 30 prosent av respondentene benyttet standardkontraktene SSA-U eller IKT 2010, mens kun en respondent benyttet PS2000.[134] Disse tallene viser klart at mange fortsatt bruker tradisjonelle kontrakter med fastsatt kravspesifikasjon til tross for at prosjektene benytter smidig programvareutvikling.

Programvareutvikling er, som jeg har påpekt tidligere, komplekst og risikoen for ikke å lykkes med prosjektene er overhengende. Som nevnt er et av formålene med denne oppgaven å bidra i arbeidet med å utforme kontrakter som kan sikre vellykket gjennomføring av smidig programvareutvikling. Det er komplisert og utfordrende å utforme rettferdige og balanserte kontrakter, og det er mange hensyn som skal tas. I arbeidet med å utfordre det tradisjonelle og etablerte utgangspunktet for tilvirkningskontrakter, bør det velkjente sitatet fra Mencken ikke undervurderes:

> *«For every complex problem, there is a solution that is simple, neat, and wrong.»*
> H. L. Mencken

3.2 Generelt om kontraktsutforming

I et tilvirkningsprosjekt skal kontrakten være et sentralt styringsdokument. Den sier noe om utvekslingen av ytelsene – et tilvirket produkt mot penger. I tillegg skal kontrakten være konfliktforebyggende, samtidig som den skal regulere forholdende når konflikter faktisk oppstår. Det er et viktig poeng at reguleringen av plikter og rettigheter skal være balansert, rettferdig og rimelig. Utgangspunktet er en beskrivelse av hva leverandøren skal lage og når det skal leveres. Dette skal balanseres med hva og når kunden skal betale. Underveis i tilvirkningsprosjektet oppstår det som regel uforutsette situasjoner som gjør at de opprinnelige pliktene og rettighetene ofte må justeres. I tillegg er det ofte behov for en viss fleksibilitet rundt hva som skal leveres, for eksempel at kunden ønsker endringer i det som opprinnelig er avtalt. Kontraktsutformingen bør derfor regulere elementer som *arbeidsbeskrivelse*, *prismodell*, *endringshåndte-*

133 ibid. s. 21
134 ibid. s. 22

ring og *konfliktløsningsmetoder*. Partenes plikter og rettigheter gir en nærmere beskrivelse av hvordan de skal forholde seg til disse elementene. Kontraktsretten handler også om *kontraktsbrudd* og eventuelt tilhørende *sanksjoner*, som er nødvendige mekanismer for å håndtere situasjoner når prosjektet går i en retning partene ikke ønsker. Kontraktsbrudd handler ikke først og fremst om skyld, men om hvordan konsekvensene av uønskede situasjoner bør fordeles mellom partene. Utgangspunktet er at den parten som er nærmest til å forebygge risikoer for at uønskede situasjoner oppstår, er den parten som også skal bære konsekvensene av disse situasjonene, se underkapittel 2.5.2.

Standardkontrakter er ofte utgangspunktet når et kontraktsforhold skal etableres, og det kan være en stor fordel å basere avtaleinngåelsen på disse. Transaksjonskostnadene kan reduseres, og man har et sett med standardvilkår som har en innbyrdes sammenheng. Spesielt på områder med «agreed documents», som for eksempel entreprise og fabrikasjon, har representanter for både leverandører og kunder forhandlet seg frem til balanserte kontrakter. For standardkontrakter som IKT 2010 og SSA-U kan det nok hevdes at de på visse områder er ubalanserte i enten leverandørens eller kundens favør.[135]

Som jeg har trukket frem i kapittel 3.1 blir standardkontrakter ofte benyttet for smidig programvareutvikling. Dette er standardkontrakter som i utgangspunktet ikke er tilpasset disse utviklingsmetodene. Mye av reguleringen i standardkontrakter skjer i bilagssettene, og det er derfor fristende å regulere den smidige utviklingsmetoden i bilagene. I tillegg er det også vanlig å ta inn endringer i standardvilkårene i bilagene. Det er heller ikke uvanlig at bilagene blir utformet av personer som ikke har full oversikt over den indre systematikken i standardvilkårene. Resultatet kan fort bli at reguleringen i bilagene og standardvilkårene henger dårlig sammen.[136] En vanlig årsak til konflikter i entreprisekontrakter er spesialtilpassede endringer i standardvilkårene, og tilhørende usikkerhet om hvordan kontrakten skal tolkes i lys av disse endringene.[137] En annen årsak til uoversiktlige kontraktsforhold er det såkalte *lag-på-lag-prinsippet*. Oppbyggingen av en standardkontrakt gjør at kontrakten med tilhørende bilag (spesielt ved en offentlig anskaffelse) består av en rekke dokumenter som delvis kan være motstridende.[138] Denne typiske strukturen til standardkontrakter gjør at det nok i mange tilfeller hadde lønt seg å gjennomføre en konsolidering av kontraktsdokumentene før kontrahering.[139] Omfattende og uoversiktlige kontraktsdokumenter harmonerer etter min mening dårlig med prinsippene bak smidig programvareutvikling. Spesielt manifestets punkt *samarbeid med kunden fremfor kontraktsforhandlinger*, tydeliggjør dette. Difis nye

135 Jf. Føyen (2006) s. 63
136 ibid. s. 66
137 Se Knag (2010) s. 88-96
138 ibid. s. 101-103
139 ibid. s. 109

avtale som er beregnet for smidig programvareutvikling, SSA-S, synes å være bygget opp på samme lest som SSA-U. Konsekvensene er at den nye avtalen kan få de samme utfordringene med en uoversiktlig kontrakt.

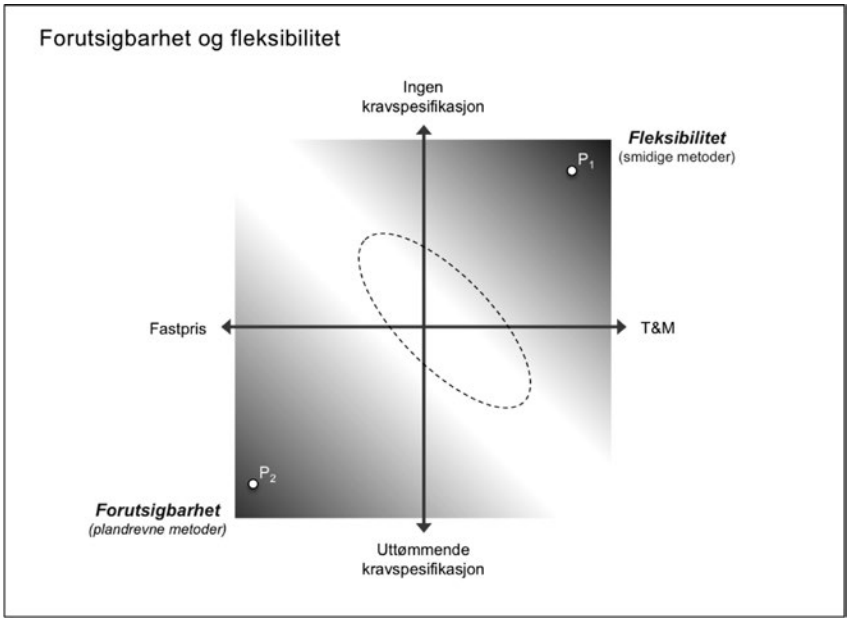
Jeg mener at kontrakter for smidig programvareutvikling må utformes på bakgrunn av tankesettet og prinsippene som ligger bak en smidig utviklingsmetode. Ved bruk at smidige metoder vil det i følge Henschel være «[…] illogical to use a form of contract where all the responsibility was put on one party only and keeping the relation on an 'arms-length'.»[140] Spesielt når kravspesifikasjonen ikke er endelig og uttømmende er det viktig at selve prosessen og utviklingsmetoden er detaljert beskrevet i kontrakten.[141] I tillegg mener jeg det er viktig at kontrakten er konsolidert og ikke mer omfattende enn nødvendig. Dette er viktig fordi kontrakten skal brukes aktivt som et styringsverktøy for alle interessenter i prosjektgjennomføringen. Kontrakten skal ikke hentes frem kun når prosjektet tar en uønsket retning. Likevel er det en balansegang mellom hvor gjennomregulert kontrakten bør være kontra en minimumskontrakt. Disse ytterpunktene illustrerer jeg i figur 1.4. Alle interessenter i et prosjekt har ikke oversikt over bakgrunnsretten og dens betydning for den konkrete kontrakten. Derfor kan det være nødvendig å ta med vilkår i kontrakten selv om bakgrunnsretten er klar nok. På andre områder er bakgrunnsrettens løsninger mindre tydelige, som for eksempel ved endringshåndtering og erstatningsansvar. Derfor bør disse forholdene reguleres i kontrakten for å skape klarhet og forutsigbarhet.

Et av de grunnleggende elementene i en smidig utviklingsmetode er *fleksibilitet*. Fleksibilitet først og fremst når det gjelder hva som skal tilvirkes. Programvareutviklingen tar ikke utgangspunkt i en uttømmende og endelig kravspesifikasjon. Kombineres dette utgangspunktet med en prismodell basert på betaling for medgått tid, såkalt *regningsarbeid*, oppnås det en høy fleksibilitet. I motsetning til dette er *forutsigbarhet* det grunnleggende utgangspunktet for plandrevene utviklingsmetoder. Forutsigbarhet når det det gjelder hva som skal lages og til hvilken pris. Ved høy fleksibilitet er det kunden som i hovedsak bærer risikoen for usikkerheten i prosjektet, men ved høy forutsigbarhet er det leverandøren som bærer risikoen. Disse ytterpunktene ved risikofordelingen har jeg illustrert i figur 1.1. Sammenhengen mellom fleksibilitet og forutsigbarhet i relasjon til utviklingsmetodene illustrer jeg figur 3.2. Jo høyere opp til høyre prosjektet plasseres, jf. P1, jo større fleksibilitet og en smidig utviklingsmetode er egnet. På samme måte oppnås større forutsigbarhet med en plandreven utviklingsmetode hvis prosjektet plasseres nede til venstre, jf. P2. Figuren viser ingen klare grenser, men med det hvite, midtre område mener jeg å vise at den smidige utviklingsmetoden mister sine grunnleggende prinsipper. På samme måte vil en plandreven utviklingsmetode være lite hensiktsmessig i dette område, fordi

140 Se Henschel (2012) s. 244

141 Jf. Føyen (2006) s. 129-130

parameterne (for eksempel pris og spesifikasjon) som brukes for å skape forutsigbarhet blir mindre statiske. Men, som jeg innledet oppgaven med, er det ofte slik at kunden ikke vil bære all risiko for usikkerheten alene, selv om kunden ønsker smidig programvareutvikling. Jeg ønsker å drøfte alternative måter å spesifisere programvare på slik at en viss forutsigbarhet kan oppnås med tanke på sluttproduktet. I tillegg skal de grunnleggende verdiene og prinsippene for smidig programvareutvikling ivaretas. Det stiplede område jeg har markert i figur 3.2 viser det området der jeg mener disse behovene er ivaretatt. Etter min mening vil utforming av kontrakter som kan benyttes av prosjekter som befinner seg innenfor det stiplede område, være et verdifullt bidrag til å lykkes med smidig programvareutvikling.



Figur 3.2: Sammenhengen mellom forutsigbarhet og fleksibilitet og utviklingsmetoder

I den videre behandlingen av kontraktsregulering av smidig programvareutvikling bruker jeg *Scrum* som den foretrukne modellen for utviklingsprosessen. Dette er fordi den har eksistert i bransjen en stund og fortløpende blitt forbedret. I tillegg er den som tidligere nevnt et rammeverk slik at den kan ivareta prosessen som helhet. Og dermed, etter min mening, egnet som utgangspunkt for kontraktsregulering. Etter denne gjennomgangen av kontraktsutforming

generelt, og smidig programvareutvikling spesielt, vil de videre drøftelsene basere seg på følgende prinsipper når det gjelder utformingen:

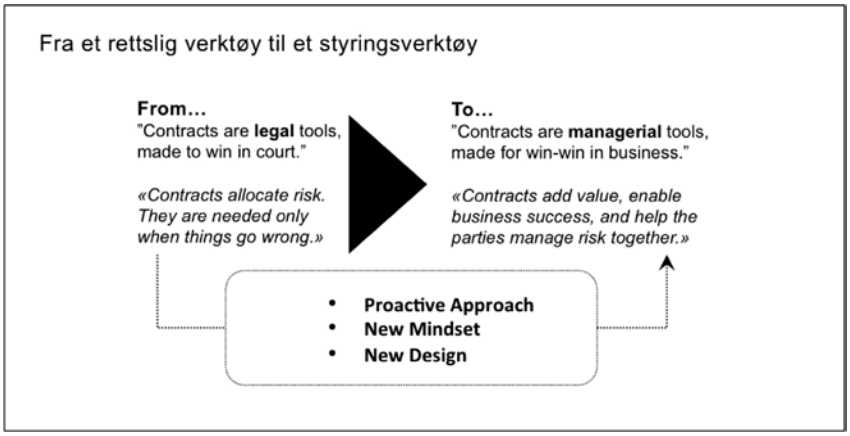
- balansere detaljregulering med kontraktens «brukervennlighet»
- balansere ønskene om fleksibilitet og forutsigbarhet
- manifestet for smidig programvareutvikling med tilhørende prinsipper
- proaktiv juss som en tilnærming til økt verdiskapning

3.2.1 Den proaktive jussens tilnærming

Ofte er utgangspunktet i kontraktsrettslige drøftelser et perspektiv *ex post*, se underkapittel 1.5.2. Dette innebærer at problemstillinger som blir behandlet er knyttet til konflikter som allerede har oppstått, og at problemstillingene drøftes fra dommerens ståsted.[142] I denne oppgaven er målsettingen at kontrakten skal kunne brukes som et viktig styringsverktøy som kan bidra til vellykket gjennomføring av smidig programvareutvikling. Kontraktsutformingen må derfor ta hensyn til både praktiske spørsmål og juridiske spørsmål for å kunne ivareta mulighetene for gevinstrealisering. Dragsted påpeker at «[k]ontrakten er et kommercielt instrument med andre formål end blot en begrænsning av juridisk risiko.»[143] Kontrakten må på best mulig måte legge til rette for at partene sammen klarer å gjennomføre prosjektet uten at regler om kontraktsbrudd og sanksjoner kommer til anvendelse. Likevel må kontrakten regulere partenes plikter og rettigheter når prosjektet går i en uønsket retning og konflikter oppstår. Jeg synes Knut Kaasen beskriver denne tosidigheten på en treffende måte: «En god kontraktsatmosfære uten rettslige holdepunkter når det røyner på, er risikable ting.»[144]

Den proaktive jussens tilnærming til kontraktsutformingen er basert på et *ex ante* perspektiv. For å vise hvordan jeg bruker proaktiv juss i utformingen trekker jeg frem en illustrasjon av Helena Haapio. Figur 3.3 viser hvordan søkelyset rettes fra kontrakten som et rettslig verktøy til et styringsverktøy. Den proaktive tilnærmingen innebærer ifølge Haapio en endret tankegang (*New Mindset)* og ny type kontraktsutforming (*New Design).*

142 Jf. Haaskjold (2013) s. 93 (petit)
143 Dragsted (2012) s. 13
144 Se Kaasen (1994) s. 31



Figur 3.3: Kontraktsutforming basert på en proaktiv tilnærming. Se Haapio (2013) s. 68

Haapios arbeid med utforming av kontrakter med en proaktiv tilnærming, tar utgangspunkt i at de tradisjonelle kontraktene ikke optimaliserer verdiskapning og øker faren for problemer og konflikter. Dette gjelder spesielt i nyere tid hvor produkter og tjenester i markedet finnes i større varianter og er mer kompliserte enn det som var tilfellet da de tradisjonelle kontraktene ble utformet. I tillegg er det mer vanlig for aktørene i markedet å legge til rette for langsiktige relasjoner, fremfor engangstransaksjoner. Et godt eksempel på dette er nettopp programvareutvikling med smidige metoder, som tas i bruk fordi de tradisjonelle metodene ikke er tilfredsstillende. De tradisjonelle kontraktene har omfattende regulering av situasjoner hvor problemer og konflikter oppstår, og legger mindre vekt på hvordan partene skal oppnå verdiskapning.[145] I følge Dragsted kan visse kontraktsmekanismer i praksis skape konflikter og hindre ønskede resultater.[146] I tillegg har kontraktene en utpreget juridisk sjargong, og vilkårene kan være vanskelig å få tak på for alle interessenter i et prosjekt. Dette gjør det utfordrende å bruke kontrakten aktivt som et styringsverktøy.[147] Også det såkalte *lag-på-lag-prinsippet*, se kapittel 3.2, er med på å gjøre kontraktene uoversiktlige.

Når kontrakten skal brukes av alle interessentene i et prosjekt, er det også ulike behov knyttet til hvordan kontrakten bør utformes. I lys av dette mener Haapio at tankegangen og utformingen av kontrakter må endres, jf. figur 3.3. Hun legger spesielt vekt på samarbeidet mellom ulike profesjoner og interessen-

145 Jf. Haapio (2013) a. 46-47
146 Jf. Dragsted (2012) s. 13
147 Jf. Haapio (2013) s. 46

ter når kontrakten skal utformes.[148] Bare på denne måten kan interessentenes ulike behov ivaretas. Den danske undersøkelsen til Dragsted og Klint underbygger dette ved at flere av respondentene ga tilbakemeldinger på «at når kontrakten er uforståelig eller ikke passer til den praktiske hverdag, bliver den uanvendelig i praksis.»[149]

Helena Haapio omtaler kontraktene med endret tankesett og en ny type kontraktsutforming for «Next Generation Contracts», og lister opp følgende egenskaper for disse: «

- be aligned and integrated with the business and its objectives
- be functional: fit for purpose and work as managerial instruments
- guide and enable the achievement of business and legal objectives
- match their business users' expectations and perceptions
- produce predictable business outcomes without negative surprises
- help build and maintain good business deals and relationships
- help balance risk with reward and minimize claims and disputes
- guide interpretation and prevent unintended interpretation
- ensure compliance with pre-existing commitments and applicable laws
- be easy to use, understand and act upon, yet legally and financially sound
- communicate their core content not only textually but also visually
- meet the criteria of "good" contracts and good documents»[150]

Det er en svært omfattende oppgave å utforme en komplett kontrakt for komplekse tilvirkningsforhold. Som punktlisten over viser er det mange egenskaper og fasetter en *Next Generation Contract* skal ha. I denne oppgaven skal jeg langt fra forsøke å utforme en komplett avtale for smidig programvareutvikling basert på proaktiv juss. Jeg vil bruke tilnærmingen for å komme et lite steg videre i arbeidet med utforming av kontrakter som øker sjansen for at prosjekter som benytter smidig programvareutvikling lykkes.

3.3 Smidige standardkontrakter

Før jeg går løs på drøftelsene av kontraktsutformingen av smidig programvareutvikling, er det nødvendig å se nærmere på noen av de få standardkontraktene som er utarbeidet for å støtte smidige metoder.

148 Se nærmere Haapio (2013) s. 52-56
149 Jf. Dragsted (2013) s. 31
150 Se Haapio (2013) s. 72

3.3.1 SSA-S

SSA-S, Statens Standardavtale – Smidigavtalen,[151] «[…] gjelder leveranse av Programvare […] utviklet ved bruk av en metode for Smidig programvareutvikling, som nærmere beskrevet i bilag 6», se pkt. 1.1. Avtalen definerer, i pkt. 17, smidig programvareutvikling som følger:

> «En samling programvareutviklingsmetoder basert på iterativ og inkrementell utvikling, hvor krav og løsninger utvikles gjennom samarbeid mellom selvstyrte team bestående av folk med forskjellige arbeidsfunksjoner ('crossfunctional')»

Kontrakten legger altså opp til at det skal brukes en eller annen smidig utviklingsmetodikk som ikke er tatt inn i standardvilkårene, men som skal angis i bilaget. I bilaget er det ikke laget noe forslag til hvilken metode som skal benyttes og hvordan denne bør beskrives. Det blir da opp til partene å lage denne beskrivelsen selv. Men kontrakten tydeliggjør at gjennomføringen er «basert på fleksibilitet i systemutviklingen og et tett samarbeid mellom Kunde og Leverandør for å realisere behovsbeskrivelsen», jf. 1.4, første avsnitt. Fleksibiliteten klargjøres i andre avsnitt med at visse presiseringer og omprioriteringer ikke skal anses som endringer som skal håndteres etter endringsprosedyrene i avtalens kapittel tre. Videre regulerer kontrakten at utviklingen skal skje med delleveranser, og at hver delleveranse skal bestå av en eller flere iterasjoner, jf. pkt. 2.1.1. Kontrakten kan brukes til både selve programvareutviklingen og til for eksempel opplæring, konvertering og rutineutvikling – såkalte Øvrige leveranselementer. Jeg går ikke nærmere inn på sistnevnte, men etter min mening er spørsmålet om slike leveranser heller bør reguleres i egen avtale.

I et konkurransegrunnlag skal kunden blant annet utarbeide «behovsbeskrivelse og krav» i bilag 1. Dette er grunnlaget for leverandørens løsningsbeskrivelse i bilag 2. Det legges opp til et tosporet opplegg for behovsbeskrivelse og krav. Det ene sporet er at kunden beskriver «formålet og de funksjonelle behov programvaren skal oppfylle», jf. veiledning i bilag 1. Det andre sporet er såkalte ikke-funksjonelle krav, som typisk gjelder krav til «pålitelighet, effektivitet, brukbarhet, vedlikeholdbarhet eller portabilitet etc», jfr. avtalens pkt. 17. Hensikten med dette tosporede opplegget er at de ikke-funksjonelle kravene skal være uttømmende og endelige. Dette fremkommer i veiledningens pkt. 3.3.2. Øvrige krav (typisk funksjonelle) skal utarbeides av både kunde og leverandør etter kontraktsinngåelsen. Disse kravene skal være en detaljering av behovsbeskrivelse og krav i bilag 1. Det er ikke noe krav om at de funksjonelle kravene skal være uttømmende spesifisert før utviklingen tar til, slik det er krav om i PS2000 Standard og PS2000 Smidig.

151 Difi (2014)

Etter mitt syn skaper det noen utfordringer å ha et tosporet system når det gjelder funksjonelle og ikke-funksjonelle krav. Kontrakten legger til rette for smidig programvareutvikling med fleksibilitet og kundens involvering knyttet til hvert inkrement, men dette gjelder kun de funksjonelle kravene. Metoden for utviklingen av de ikke-funksjonelle kravene ligger nærmere en fossefallsmodell enn en smidig utviklingsmetode. Forskjellen ligger i at kontrakten pålegger delleveranser også for ikke-funksjonelle krav. For det første fører det tosporete systemet til at kontraktsvilkårene må anvendes ulikt på ulike deler av programvaren ved hver delleveranse. Dette skaper en uoversiktlighet som gjør det utfordrende å forholde seg til vilkårene. For det andre er det ofte ikke et klart skille mellom funksjonelle og ikke-funksjonelle krav. Dessuten vil begge kravtypene påvirke hverandre og være avhengige av hverandre. Da blir det utfordrende med en fleksibilitet for noen krav som likevel må forholde seg til statiske og endelige krav. Disse motsetningen er til en viss grad regulert i avtalens pkt. 2.3.1, andre avsnitt. Her pålegges leverandøren en rådgivnings- og opplysningsplikt overfor kunden når det gjelder håndtering av forholdet mellom ikke-funksjonelle og funksjonelle krav. Hvis kunden, til tross for leverandørens råd, ønsker utvikling av et funksjonelt krav som hindrer oppfyllelse av ikke-funksjonelle krav, må dette håndteres som en endring etter avtalens kapittel tre.

Når det gjelder prismodell tar SSA-S utgangspunkt at kunden betaler for medgåtte timer, regnet i en løpende *teampris*, jf. bilag 7 og avtalens veiledning kapittel 5.1. I tillegg er det lagt til rette for en bonusordning knyttet til omfang og kvalitet, tid og pris. Både bonusordningen og andre vilkår i kontrakten som blant annet gjelder endringshåndtering, dagbøter og erstatning er relatert til begrepet *Estimert Totalkost*. Denne er definert som «[e]stimert totalpris for kontrakten. Tilsvarer kontraktspris i øvrige SSA-er», jf. avtalens pkt. 17. Selv om de funksjonelle kravene skal utarbeides etter kontraktsinngåelsen, forutsettes likevel at leverandøren har estimert en totalkostnad. Jeg har tidligere vært inne på utfordringene med å estimere programvare på grunn av kompleksiteten og usikkerheten. Spørsmålet blir da hvor reell denne estimering faktisk har vært. Er den ikke reell blir det utfordrende å forholde seg til kontraktsvilkårene som bruker estimatet.

Etter min mening er kontrakten relativt universell. Den skal kunne støtte ulike smidig utviklingsmetoder, men også støtte utvikling av programvare (ikke-funksjonelle krav) basert på endelig og uttømmende kravspesifikasjon. Kunden er pålagt større ansvar i denne avtalen enn i SSA-U, fordi kunden skal medvirke til gjennomføringen i henhold til den utviklingsmetoden som blir spesifisert i bilag 6, jf. avtalens pkt. 6.1. Når det gjelder leverandørens ansvar for ytelsen har pkt. 5.1, første avsnitt følgende tekst:

> «Leverandøren har ansvar for at leveransen dekker behovs- og løsningsbeskrivelsen i bilag 1 og 2 med eventuelle endringer i bilag 9, likevel slik at detaljspesifikasjonen og godkjenningskriteriene skal utgjøre den endelige spesifiseringen av hva som skal leveres for de områdene den dekker.»

Leverandøren har her et resultatansvar for at programvaren dekker det som er spesifisert i bilag 1 og 2. Hva dette innebærer må avgjøres i den konkrete avtalen, men jo mer detaljert og tydelig spesifikasjonen er jo tydeligere fremstår ansvaret som en resultatforpliktelse. De ikke-funksjonelle kravene vil som nevnt være uttømmende spesifisert. Siste delen av setningen synes å være en prioritetsregel slik at detaljspesifiseringen som kunden og leverandøren samarbeider om gjelder foran det som er spesifisert i bilag 1 og 2. Leverandørens resultatansvar dekker også disse spesifikasjonene. Denne bestemmelsen må tolkes i lys av hvilken utviklingsmetode som spesifiseres i bilag 6, og hvordan denne metoden beskriver arbeidsdelingen og ansvaret mellom kunden og leverandøren.

3.3.2 PS2000 SOL

PS2000 SOL er DNDs kontraktsstandard for oppdragsbasert smidig leveranse. Avtalen er utarbeidet for prosjekter som benytter smidig utviklingsmetode, uten en uttømmende kravspesifikasjon før utviklingen starter. Dette i motsetning til for eksempel PS2000 Smidig, som forutsetter at det er utarbeidet en slik kravspesifikasjon før oppstart av iterasjonene, se avsnitt 2.5.1.1. Avtalen forutsetter at partene avtaler en bistandsavtale og en eller flere oppdragsavtaler som gjelder parallelt, pkt. 1.5. Det avtales en samlet kapasitet på ressurser som leverandøren skal stille til disposisjon, og som kunden er forpliktet til å benytte, jf. pkt. 1.4. Bistandsavtalen skal dekke alt arbeid leverandøren gjør som «[…] ikke er direkte knyttet til utviklingen av selve Programvaren», jf. pkt. 1.5.1. Dette er arbeid hvor leverandøren bistår kunden med blant annet utarbeidelse av behovsanalyser og løsningsbeskrivelser. Behovsanalysen består av epos og brukerhistorier,[152] som danner grunnlaget for produktkøen. Begge parter utarbeider en løsningsbeskrivelse, som er en detaljspesifisering med estimater, jf. pkt. 5.3.

Partene inngår en eller flere oppdragsavtaler, og hver oppdragsavtale skal dekke selve programvareutviklingen («Konstruksjon»), jf. pkt. 5.4. Konstruksjonen dekker en eller flere iterasjoner, jf. pkt. 5.5. En eller flere iterasjon er en «Leveranse», som kunden skal verifisere ved en godkjenningsprøve, jf. pkt. 5.6. Vederlaget skal for bistandsavtalen i utgangspunktet være basert på løpende timer. For oppdragsavtalen er utgangspunktet løpende timer eller målpris, jf. pkt. 6.1.

152 Se avsnitt 2.4.1.1 om epost og brukerhistorier.

Hvis det er avtalt at leverandøren har «[…] ansvaret for overordnet ledelse og styring av Løsningsbeskrivelsen […]», er det kontraktsbrudd hvis løsningsbeskrivelsen er forsinket eller er mangelfull, jf. pkt. 10.1.1. Dette fører til at kunden ikke er forpliktet til å utnytte avtalt kapasitet, samt at kunden kan kreve erstatning hvis det forsinker oppstart av konstruksjonen. Videre er en leveranse forsinket hvis den ikke oppfyller godkjenningskriterier under godkjenningsprøven. Dette utløser dagbøter, jf. pkt. 10.1.2. Det foreligger også kontraktsbrudd hvis faktisk timeforbruk overstiger estimatet med det dobbelte, jf. 10.1.3. Ellers er kundens medvirkningsplikt tydeliggjort i pkt. 10.2.2.

3.3.3 Utenlandske

I dette underkapittelet forklarer jeg kort om to utenlandske standardkontrakter som er interessante med tanke på smidig programvareutvikling. Den første er den nye danske K03 som er beregnet for prosjekter med smidige utviklingsmetoder.[153] Den andre, som kalles *Flexible Contracts,* er basert på et initiativ fra Benefield og Atkinson.[154]

K03

Den danske standardavtalen K03 er utgitt av Digitaliseringsstyrelsen i samarbeid med Kammeradvokaten. It-bransjen har deltatt i arbeidsgrupper. Kontrakten er beregnet på prosjekter som benytter smidige utviklingsmetoder, men den legger opp til at tid og pris ligger fast. Fleksibiliteten er knyttet til hva som skal ytes.[155] Beskrivelsen av ytelsen skal ved kontraktsinngåelsen være spesifisert i form av overordnede krav, som tar utgangspunkt i kundens behov, jf. bilag 3. Kundens behov er omtalt som «Forretningsmæssige Mål og Behov», jfr. pkt. 1. Det forutsettes i bilag 3a.i at kunden har gjennomført en forhåndsanalyse og utarbeidet et såkalt *business case*. Detaljeringen av kravene skjer fortløpende i forbindelse med hver iterasjon. Kontrakten skiller mellom absolutte krav og øvrige krav. Leverandøren har en resultatforpliktelse til de absolutte kravene og en innsatsforpliktelse for de øvrige kravene, jf. pkt. 3.2.2, tredje avsnitt. Ellers regulerer kontrakten det tette samarbeidet mellom partene, blant annet med en eksplisitt lojalitetsforpliktelse i pkt. 8.1.

Jeg er usikker på hvor hensiktsmessig det er å skille mellom absolutte og øvrige krav. Faren er at kunden spesifiserer mange, og kanskje detaljerte, absolutte krav. Dette vil medføre at kontrakten får mer preg av en fastpriskontrakt, med fast tid og uttømmende kravspesifikasjon. Likevel legger kontrakten opp til et tett samarbeid og har prosedyrer for endringshåndtering. Det som er interessant

153 Se Digitaliseringsstyrelsen (2012)
154 Se Benefield (2013)
155 Jf. Holsøe (2013) s. 32

er at kontrakten vektlegger kundens forretningsmessige mål og behov, som skal være styrende for gjennomføringen.

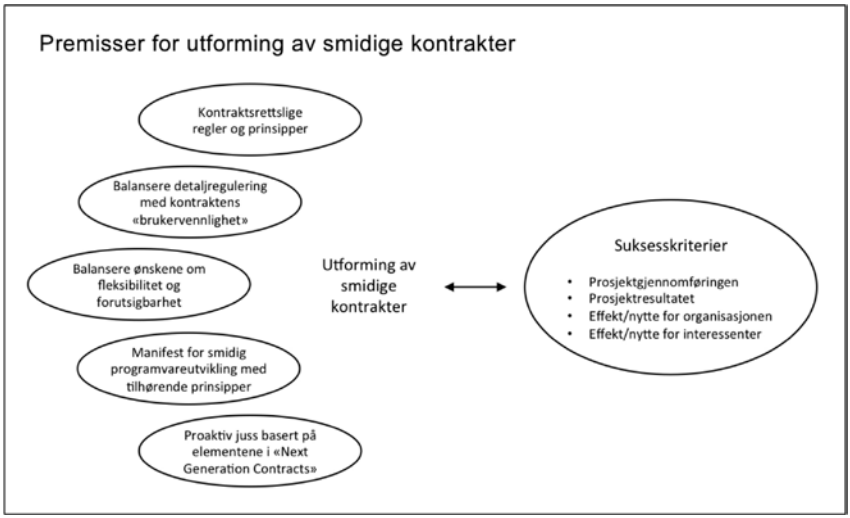
Flexible Contracts
Flexible Contracts skal fungere som et kontraktsrammeverk, hvor mange vilkår må tilpasses det enkelte prosjekt. Det som imidlertid skiller kontrakten fra tradisjonelle kontrakter er bruken av de såkalte «statements of target outcomes» (SOTO). Det forutsettes av utviklingen gjennomføres med en iterativ utviklingsmetode. Hver delleveranse skal måles på i hvilken grad man har nådd de enkelte målbare målene («target outcomes»). Disse målene tar utgangspunkt i kundens behov og ønske om gevinstrealisering. Vilkår knyttet til betaling, levering og kvalitet avtales særskilt for hver SOTO. Partene kan si opp avtalen etter hver SOTO. Selv om kontrakten er noe overordnet, er tankegangen med spesifikasjon av målbare mål interessant. Prosjektgjennomføringen styres med utgangspunktet i disse målene, mens leverandøren har frihet til å velge hva som skal utvikles slik at målene kan realiseres. Konseptet i denne kontrakten likner, etter min mening, på tankegangen bak PS2000 SOL.

3.4 Utforming av en smidig kontrakt

I en plandreven metode, som fossefallsmodellen, med uttømmende og endelig kravspesifikasjon, er det kravene som er styrende for gjennomføringen i alle fasene.[156] Dette endelige definerte omfanget danner sammen med fast pris og fastsatt leveringstid, låste rammer for prosjektgjennomføringen. De er dessuten grunnleggende vilkår i de tradisjonelle kontraktene. I tillegg utgjør disse rammene suksesskriteriene for om prosjektet blir vurdert som vellykket eller ikke.[157] I underkapittel 1.4.3 presenterte jeg et utvidet syn på suksesskriterier for prosjekter generelt og for it-prosjekter spesielt. De grunnleggende prinsippene som ligger bak smidig programvareutvikling er utformet blant annet med tanke på at utviklingsmetoden skal øke sjansen for at prosjektene lykkes. Derfor må kontraktsutformingen for disse prosjektene også ta hensyn til det utvidede synet på suksesskriterier.

I tillegg til suksesskriteriene har jeg trukket frem en rekke forhold som jeg mener det må tas hensyn til når smidige kontrakter skal utformes. I figur 3.4 illustrerer jeg dette. Til venstre i figuren har jeg listet opp disse forholdene, og til høyre illustrerer jeg at suksesskriteriene også må trekkes inn i utformingen. I tillegg brukes suksesskriteriene til å vurdere prosjektets vellykkethet både underveis og i etterkant av prosjektet.

156 Jf. Langemark (2009) s. 6
157 Se definisjon av å *lykkes* i underkapittel 1.4.3.



Figur 3.4: Premisser for utforming av smidige kontrakter

Utgangspunktet ved bruk av smidige utviklingsmetoder er at løsningen som skal lages ikke kan og skal spesifiseres endelig og uttømmende ved kontraktsinngåelsen. Da blir spørsmålet: Hvordan skal kontraktsgjenstanden spesifiseres med tanke på risikofordeling og et vellykket resultat? Svaret på dette ligger i oppgavens problemstilling, som handler om alternative måter å spesifisere kontraktsgjenstanden på. Min tilnærming er at *spesifikasjonen må baseres på et sett av suksesskriterier som er unikt for hvert enkelte prosjekt*, og som er tett knyttet til gevinstrealisering.

3.4.1 Gevinstrealisering

Hvilke suksesskriterier som ligger til grunn for hvert enkelte prosjekt vil variere fordi det ligger i prosjektets natur at alle prosjekt er unike. Erkjennelsen av at prosjektene ikke kun kan måles på omfang, tid og kostnad, har resultert i en rekke metoder, teorier, retningslinjer og veiledere.[158] De handler om hvordan man skal definere suksesskriterier for et prosjekt, samt hvordan disse kriteriene skal brukes som et styringsverktøy både før et prosjekt starter opp og underveis i prosjektgjennomføringen. Disse fremgangsmåtene handler om *gevinstrealisering.* Gevinstrealisering har tre trinn: planlegge gevinstrealisering, gjennomføre prosjektet med fokus på gevinstene og til slutt følge opp med realisering av gevinstene.[159]

158 Se for eksempel veilederen fra Senter for statlig økonomistyring (2010)
159 Jf. Karlsen (2013) s. 495

Et prosjekt gjennomføres aldri for prosjektets skyld, men for å oppnå en endring fra situasjonen før prosjektet ble igangsatt. I tillegg skal endringene gi gevinster i en eller annen form. Gevinstrealisering gjør at prosjektene endres fra å ha et leveranseperspektiv til å ha et verdi- og nytteperspektiv. Smidig programvareutvikling skal bidra til dette perspektivet, og det kommer tydelig frem i manifestets første prinsippet:

> «Vår høyeste prioritet er å tilfredsstille kunden gjennom tidlige og kontinuerlige leveranser av programvare som har verdi»[160]

I tillegg til at smidig programvareutvikling har et prinsipp om å levere programvare som har verdi, er det også slik at verdien og nytten skal kunne leveres fortløpende ved «tidlige og kontinuerlige leveranser». Dette gjør at det er mulig å måle og ta ut gevinster også *før* prosjektet avsluttes, i motsetning til ved plandreven utvikling. I tillegg kan prosjektet justere utviklingen basert på erfaringene med gevinstrealiseringen underveis. Prosjektet for pensjonsreformen i Statens pensjonskasse (PERFORM) benyttet en iterativ utviklingsmetode basert på en PS2000-kontrakt. Næss og Frøyland har pekt på at delleveransene underveis i prosjektet ble målt på gevinstrealisering, og justeringer basert på målingene kunne legges inn i neste delleveranse.[161] Dette viser viktigheten av at kontrakten bør inneholde styringsinformasjon med tanke på gevinstrealisering.

De fleste prosjekter blir satt i gang på bakgrunn av ønske om endringer og forventninger til gevinster. Men det er ikke uvanlig at de forventede gevinstene uteblir, og det kan være mange årsaker til det. Karlsen[162] lister opp ulike årsaker som:

- fravær av fokus på gevinster i prosjektgjennomføringen
- prosjektmedlemmer har ikke et felles bilde av gevinstmulighetene
- mangel på kommunikasjon og planlegging gjør at ikke alle styrer mot de samme målene og har den samme oppfatningen av hvorfor prosjektet gjennomføres.

Det er mange faktorer som er viktige for å lykkes med gevinstrealisering, men jeg vil trekke frem to:[163]

- klare for mål for gevinster, og målene skal beskrive de effekter man ønsker å oppnå og forklarer dermed hvorfor prosjektet gjennomføres – dette er *effektmål*.
- utarbeidelse av en *gevinstrealiseringsplan* før prosjektet settes i gang.

160 Se vedlegg 5.1
161 Se nærmere Næss (2012) s. 129
162 Se Karlsen (2013) s. 495
163 ibid. s. 496-497

Disse faktorene henger nøye sammen med årsaken til at de forventede gevinstene uteblir. Derfor blir det, etter min mening, viktig å se nærmere på reguleringen av disse to forholdene i en kontrakt. Jeg vil nå gå nærmere inn på gevinstrealiseringsplanen og effektmål.

3.4.1.1 Gevinstrealiseringsplanen

Før en gevinstrealiseringsplan utarbeides må det gjøres et arbeid internt i kundens organisasjon. Arbeidet består av blant annet å analysere og kartlegge hvorfor et prosjekt skal gjennomføres, hva hensikten er, hvem som blir berørt og hvilke gevinster som kan oppnås. Dette saksgrunnlaget for en gevinstrealiseringsplan kalles ofte et *business case*. I et offentlig prosjekt starter gjerne prosessen med en samfunnsøkonomisk analyse.[164] En viktig del av forhåndsanalysen er å kartlegge hvilke personer og grupper som blir berørt, og som er viktige for å gjennomføre endringene og realisere gevinstene.[165] Gevinstrealiseringsplanen utarbeides på bakgrunn av forhåndsanalysen, og skal vise sammenhengen mellom tiltak og gevinster, i tillegg til hvilke målsettinger gevinstene understøtter. Videre sier den noe om hvordan gevinstene skal måles. Gevinstene med tilhørende måleindikatorer vil være en del av prosjektets suksesskriterier. Gevinstrealiseringsplanen brukes aktivt både underveis i prosjektgjennomføringen og etter at prosjektet avsluttes. Planen er dynamisk og nye gevinstmuligheter, som oppdages underveis i prosjektgjennomføringen, kan tas inn i planen. Som nevnt finnes det flere metoder for gevinstrealisering, og beskrivelsen over er basert på den britiske *Benefits Management Modell* (BMM), som har vært til inspirasjon for de norske modellene og veilederne.[166]

Gevinstrealiseringsplanen er også et viktig grunnlag for et *prosjektmandat*.[167] Prosjektmandatet er dokumentet som definerer et prosjekt, og er en avtale mellom toppledelse/linjeledelse og prosjektleder.[168] Dokumentet inneholder blant annet elementer som:

- Bakgrunnen for prosjektet
- Prosjektets ulike mål (effekt, resultat)
- Rammebetingelser (interne og eksterne)
- Roller og ansvar
- Omfang, tidsplan og budsjett

Hvis prosjektet skal gjennomføres med en ekstern leverandør, blir elementer fra prosjektmandatet gjenspeilet i kontrakten mellom kunden og leverandøren.

164 Jf. Senter for statlig økonomistyring (2010) s. 20
165 Jr. Hellang (2012) s. 53
166 ibid. s. 51
167 Se Karlsen (2013) s. 96-97
168 ibid. s. 84

Roller og ansvar blir kontraktsfestet, kanskje sammen med ulike rammebetingelser som er relevante for leverandøren. Hvis prosjektet skal gjennomføres med en plandreven metode er, som tidligere nevnt, elementer som omfang, tidsplan og budsjett sentrale.

Uansett om et prosjekt skal gjennomføres med en plandreven eller smidig utviklingsmetode, er det hensiktsmessig å ta inn elementer fra gevinstrealiseringsplanen i kontrakten. Ifølge Dragsted er kontraktens formål å «understøtte realisering af Kundens business case og de gevinster, der er identificeret i den grundlæggende gevinstrealiseringsplan.»[169] I en undersøkelse kommer det frem at respondentene til en viss grad bruker metoder for å utarbeide gevinstrealiseringsplaner, men at planen i liten grad benyttes i kontraktsutformingen.[170] I Norge avdekker lignende undersøkelser at halvparten av respondentene i privat sektor ikke benytter gevinstrealisering,[171] mens for offentlig sektor er tallet 65 prosent.[172] I de samme undersøkelsene kommer det imidlertid frem at respondentene i stor grad vil ta i bruk gevinstrealisering for sine prosjekter i løpet av de neste tre årene. Dette understreker viktigheten av gevinstrealiseringsplaner og at disse også må få plass i kontraktene. I den danske standardkontrakten K03 kommer viktigheten av en plan for gevinstrealisering frem i bilag 3a.i. I kontrakten forutsettes det at kunden har utarbeidet et *business case*, som skal ligge til grunn for spesifikasjonen av kundens «Forretningsmæssige Mål og Behov».

3.4.2 Effektmål

En sentral del av gevinstrealiseringsplanen er *effektmålene*. Effektmål kan likestilles med formål. På engelsk brukes ofte betegnelsen «objective». Formål blir mindre brukt som betegnelse på disse målene når det handler om gevinstrealiseringen. En årsak kan være at ordet *effekt* tydeliggjør at måloppnåelse skal ha en effekt, og videre at denne effekten skal være målbar. Ordet formål blir nok brukt i flere sammenhenger og med mer generelle betydninger. Eksempler på dette i et it-prosjekt kan være at:

- løsningen som skal utvikles skal være enklere å bruke enn eksisterende system.
- løsningen skal være brukervennlig og effektiv å bruke
- løsningen skal sikre demokratiske rettigheter

Slike formålsbeskrivelser bærer mer *preg av ambisjoner*, og er *lite egnet* til å styre programvareutviklingen.[173] Likevel er det ikke uvanlig å ta inn formåls-

169 Jf. Dragsted (2012) s. 15
170 Se Dragsted (2013) s. 38
171 Rambøll Management Consulting AS (2013b) s. 64
172 Rambøll Management Consulting AS (2013a) s.71
173 Jf. Ottersten (2004) s. 23

beskrivelser i kontrakter, og angivelsen av anskaffelsens formål kan ha betydning for en mangelsvurdering.[174] Dette gjelder spesielt for plandreven utvikling hvor kontrakten inneholder en uttømmende og endelig spesifikasjon. Da kan ikke leverandøren levere funksjonalitet som åpenbart ikke er i tråd med anskaffelsens formål.[175] På en annen side kan formålsbeskrivelser som ikke er dekkende for kundens behov, være en ulempe for kunden i en konfliktsituasjon.[176] Selv om kontrakten inneholder mer eller mindre tydelige formålsbeskrivelser, kan det være vanskelig å se sammenhengen mellom en gevinstrealiseringsplan og innholdet i en kontrakt.[177] Sammenhengen mellom kontraktens formålsbeskrivelser og prosjektets leveranser vil kun komme til syne i en konfliktsituasjon, og ved en eventuell mangelsvurdering. De forholdene jeg har trukket frem her viser viktigheten av å ha klare og tydelige, og helst målbare, formålsbeskrivelser i kontrakten. I den videre fremstillingen bruker jeg derfor betegnelsen *effektmål*. Eksempler på effektmål kan være:

- Gjennomsnittlig saksbehandlingstid skal reduseres med 30 prosent.
- Antall henvendelser pr. telefon til brukerstøtten skal reduseres med 50 prosent.
- Nettbutikkens salg skal øke med 20 prosent.
- Redusere leveringstiden på bestillinger med 2 dager.
- Redusere bedriftens rentetap på utestående fordringer med 20 prosent.

Disse effektmålene er konkrete og målbare, men de er likevel overordnede slik at det finnes flere muligheter med tanke på måloppnåelse. I vurderingen av hvilke muligheter som skal utnyttes, må en ta innover seg at effekten av en valgt løsning alltid vil oppstå på individnivå.[178] Individene kan være brukerne av løsningen eller de som skal drifte, forvalte og videreutvikle løsningen. Derfor er kartleggingen av interessentene i forbindelse med utarbeidelse av gevinstrealiseringsplanen svært viktig. Ottersten og Balic omtaler interessentens forhold til løsningen som en sammenheng mellom *anvendelsesgrad*, løsningens kvalitet og nytte/effekt.[179] For eksempel kan en investering i et nytt system være beregnet til å gi en årlig gevinst på 10 millioner kr per år. Ulike utfall av denne investeringen kan tenkes. Hvis systemet blir utviklet med høy kvalitet (0,9), men at brukerne av ulike årsaker ikke anvender systemet i så stor grad som planlagt (0,2), blir nytten av investeringen 0,18 x 10 millioner kr. Et annet senario er at prosjektet ikke blir utviklet med planlagt kvalitet (0,6), men fordi man har god oversikt

174 Jf. Haaskjold (2013) s. 487-489
175 Jf. Torvund (1997) s. 74
176 l.c.
177 Jf. Dragsted (2012) s. 15
178 Jf. Ottersten (2004) s. 35.
179 Se nærmere Ottersten (2004) s. 36-37

over hvilke behov som er viktigst, blir anvendelsesgraden høyere en forventet (1,2) og nytten av investeringen kan beregnes til 0,72 x 10 millioner kr. Jeg synes dette gir et godt bilde på sammenhengen mellom suksesskriteriene som er knyttet til prosjektgjennomføringen (tid, kost, omfang/kvalitet) og prosjektresultatet (kvalitet) og de øvrige kriteriene som gjelder effekt og nytte.[180] Denne sammenhengen understreker også betydningen av å ha tydelige og målbare effektmål, som er styrende for prosjektgjennomføringen, og følgelig et viktig element i kontrakten.

3.5 Spesifikasjon av kontraktsgjenstanden

3.5.1 Innledning

Som tidligere drøftet er det svært vanskelig å utarbeide stabile, uttømmende og endelige kravspesifikasjoner, som skal ligge til grunn for prosjektets programvareutvikling. Likevel hevdes det at det er konfliktforebyggende med kontrakter som inneholder fullstendige kravspesifikasjoner.[181] Som vi har sett ligger det i programvareutviklingens natur at endringsbehovene er store. Forsøk på å utarbeide en fullstendig kravspesifikasjon ender som oftest med betydelig endringshåndtering i henhold til formaliserte prosedyrer med preklusive frister. Endringshåndteringen kan også føre til et økt konfliktnivå – spesielt hvis de endringsprosedyrene ikke følges som beskrevet i kontrakten.

Uansett hva som skal tilvirkes eller hvordan dette skal gjøres må kontrakten inneholde en eller annen form for beskrivelse av ytelsen som skal presteres. Hvis ikke det foreligger en slik beskrivelse kan det hevdes at en bindende avtale faktisk ikke er inngått.[182] Andre punkter i kontrakten kan stort sett utfylles av bakgrunnsretten. Selv om leverandøren kun skal levere ressurser til et utviklingsprosjekt må kontrakten inneholde en minimumsbeskrivelse av for eksempel hvor mange konsulenter, tilgjengelighet og kompetanse. Som jeg har vist tidligere i figur 1.1 finnes det kontrakter som ligger i spenningsforholdet mellom de tradisjonelle kontraktene, som baseres på et resultatansvar for leverandøren, og de kontraktene hvor kunden tar risikoen for det som skal tilvirkes og leverandøren har en innsatsforpliktelse.

Jeg vil nå foreslå en alternativ måte å spesifisere kontraktsgjenstanden på. Hensikten er at både kunden og leverandøren får en viss forutsigbarhet om hva som skal lages, samtidig som spesifikasjonen gir partene høy grad av fleksibilitet.

180 Se underkapittel 1.4.3
181 Jf. Føyen (2006) s. 238
182 Jf. Torvund (1997) s. 64

3.5.2 Kontraktsregulering av effektmål

Et prosjekt bør gjennomføres slik at resultatene er egnet til å realisere planlagte gevinster, og effektmålene er en viktig del av gevinstrealiseringsplanen. Kontrakten blir derfor et sentralt styringsverktøy for at prosjektresultatene faktisk gir verdi for kunden. I følge Dragsted er det ikke alltid en tydelig sammenhengen mellom kontraktens regulering og ønskede gevinster.[183] Det nærmeste man kommer er etter min mening angivelse av anskaffelsens formål i kontraktene, men som vi har sett er ikke disse formålene alltid egnet til å være styrende for prosjektresultatet. I tillegg hevdes det at leverandører i forhandlinger ønsker slike formålsangivelser ut av kontrakten.[184] Ved inngåelsen av en kontrakt med fastlagt pris og tid, samt uttømmende kravspesifikasjon, er det forståelig at formålsangivelser kan føre til en ekstra risiko for leverandørene. På en annen side kan gjennomtenkte, tydelige og klare formålsbeskrivelser være en hjelp for leverandøren i utviklingsarbeidet med tanke på utformingen av løsningens funksjoner. Dessuten kan formålsbeskrivelser øke kundens forutsigbarhet med tanke på prosjektresultatets egnethet for realisering av ønskede gevinster. Dragsted tar til orde for kontraktsregulering av en såkalt *mapping* mellom effektmål i gevinstrealiseringsplanen og kravspesifikasjonen.[185] En slik mapping innebærer at kravene til det som skal utvikles har en kobling til gevinster i gevinstrealiseringsplanen. Dette stiller store krav til både kundens og leverandørens arbeid med kravspesifikasjonen. Mappingen øker også informasjonsnivået i kravspesifikasjonen, og gjør at prioriteringen underveis i gjennomføringen kan baseres på nødvendigheten av kravet med tanke på gevinstrealisering. En slik regulering er ikke knyttet spesielt til smidige kontrakter.

Viktigheten av at prosjektresultatene skal ha verdi for kunden er, som vi har sett, et av prinsippene bak smidig programvareutvikling. For å få til dette må det ligge til grunn en slags «kobling» mellom effektmålene og det som skal utvikles. De smidige utviklingsmetodene er ikke klare på hvordan denne koblingen skal foretas. Hvordan vet vi hva som er «[..] programvare som har verdi»[186]? Ved smidig programvareutvikling forutsettes det også at det gjøres et arbeid før utviklingen kan starte, slik at man vet hvilke gevinster som ønskes. Bruk av epos og brukerhistorier[187] gir en viss hjelp med å definere verdi, men metoden har ikke noe klar kobling mellom det som angis i brukerhistoriene og gevinstrealisering. Standardkontrakten PS2000 SOL legger vekt på bruk av epos og brukerhistorier som utgangspunkt for utviklingen. Standardens veiledning understreker viktig-

183 Se Dragsted (2012) s. 15
184 Jf. Føyen (2006) s. 261
185 Se nærmere Dragsted (2012) s. 15
186 Se vedlegg 5.1
187 Se nærmere i avsnitt 2.4.1.1

heten av at epos og brukerhistorier er forankret i effekt- og resultatmål, og at disse overordnede behovsbeskrivelsene bør foreligge ved kontraktsinngåelsen.[188]

De smidige utviklingsmetodenes mangel på kobling mellom hva som skal tilvirkes og gevinstrealiseringsplanen, spesielt ved utvikling av større løsninger, er kritisert av flere eksperter. Ambler og Lines mener at prinsippene bak smidig programvareutvikling bør justeres i prosjekter hvor det utvikles større forretningssystemer.[189] Justeringene gjør at fokus endres, slik at det er *løsninger* som skal utvikles og ikke bare programvare. Videre mener de at det må samarbeides med *interessenter*, og ikke bare kunden. Definisjon av hva en kunde er kan være noe uklart, og ved å trekke inn interessentene bygger de på viktigheten av at gevinstene og nytten oppstår hos målgruppene, jf. drøftelsen i underkapittel 3.4.2. En annen viktig justering er prinsippet om «fungerende programvare er det primære målet på fremdrift» som de endrer til (min oversettelse):

> «Målbare gevinster er det primære målet på fremdrift»[190]

Når det gjelder kritikken av de smidige prinsippene, vil jeg også trekke frem Tom Gilb. Som tidligere nevnt utviklet han allerede i 1979 en iterativ og evolusjonær utviklingsmetode.[191] I denne metoden er måling av gevinstoppnåelse essensielt, og han har siden den tid utviklet metoder for dette. Metoden vektlegger viktigheten av å ta hensyn til *interessentene*, og samtidig utarbeide målbare suksesskriterier.[192] Når det gjelder kritikken av prinsippene for smidig programvareutvikling trekker også han frem at begrepet «kunde» blir for enkelt, og at det er nødvendig å «[…] identify and deal with the top few dozen critical stakeholders of your system.»[193] I samme artikkel legger han vekt på at de smidige prinsippenes bruk av *verdi* må konkretiseres og ta utgangspunkt i gevinstrealisering, samt at gevinstene og nytten for interessentene må være målbare.

Mitt utgangspunkt etter dette er at en kravspesifikasjon som beskriver funksjonelle og ikke-funksjonelle krav *ikke* kan ligge til grunn ved kontraktsinngåelse når smidig programvareutvikling skal benyttes. Slike kravspesifikasjoner vil ikke harmonere med prinsippene for smidige utviklingsmetoder, og hindre nødvendig fleksibilitet i prosjektgjennomføringen. Drøftelsene knyttet til effektmål viser at det er nødvendig at disse er tatt inn i kontrakten ved kontraktsinngåelsen. Effektmålene må imidlertid være konkrete, målbare og ha en tydelig relasjon til interessentene.

188 Jf. Den Norske Dataforening (2013b) s. 5
189 Se Ambler (2012) s. 27 flg.
190 ibid. s. 31
191 Jf. kapittel 2.4
192 Se nærmere Gilb (2005) s. 38-40
193 Jf. Gilb (2010) s. 15

Det finnes flere metoder for å utarbeide og presentere effektmål. I Tom Gilbs evolusjonære utviklingsmetode er utarbeidelse av effektmål en vesentlig del av metoden.[194] Det ligger utenfor rammene av denne oppgaven å gå nærmere inn Gilbs metode. Gojko Adzic har utarbeidet en forenklet metode han kaller *impact mapping*.[195] Denne metoden bygger blant annet på Gilbs metode. Jeg har imidlertid valgt å ta utgangspunkt i metoden til Ingrid Ottersten og Mijo Balic, som de kaller *effektstyring av IT*.[196] Jeg har valgt denne metoden fordi den etter min mening er best egnet til å illustrere hvordan effektmålene kan ligge til grunn i kontrakten som spesifikasjon av kontraktsgjenstanden, og styringen av prosjektgjennomføringen. Metodens tilnærming passer også godt sammen med en gevinstrealiseringsplan og bruk av epos og brukerhistorier. Kjernen i metoden er utarbeidelse av et visuelt *effektkart*, som også kan presenteres i skriftlig form. I tillegg er det mulig å bygge videre på effektkartet med elementer fra Gilbs metoder, avhengig av behovet i den konkrete kontrakten. Definisjonen av et effektkart er som følger:

> «Effektkarta är en metod för att skapa en beskrivning av de önskade effekterna som direkt går att använda för design av IT-produkten och för effektstyrning av IT-projektet.»[197]

3.5.3 Effektkart som ytelsesbeskrivelse

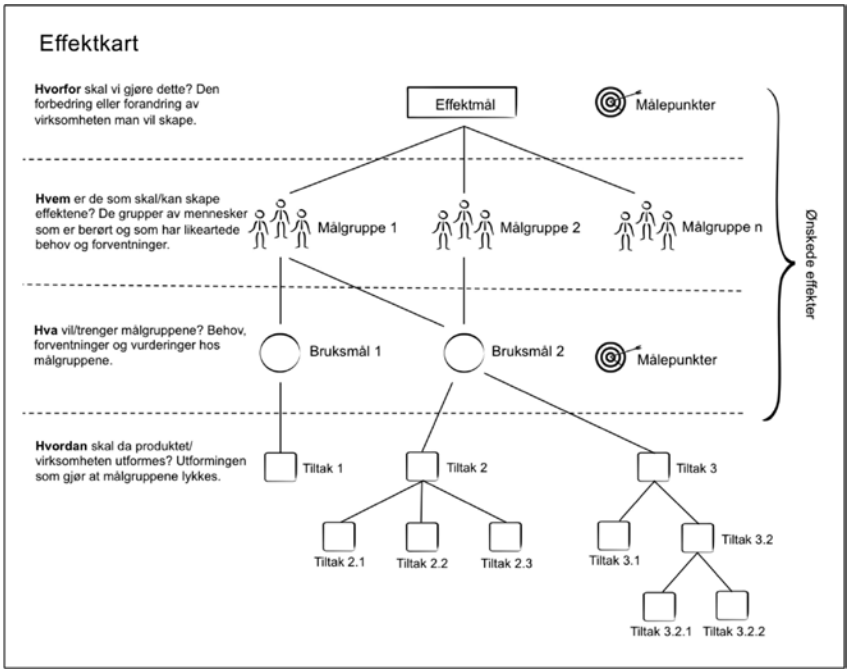
Et effektkart kan sees på som en kravspesifikasjon av hvilke elementer som påvirker de ønskede effektene. Effektmålene hentes fra gevinstrealiseringsplanen sammen med målgruppene (interessentene). Hvis det er et saksbehandlingssystem som skal utvikles kan for eksempel et effektmål være: *Gjennomsnittlig saksbehandlingstid skal reduseres med 30 prosent.* For å få til dette analyseres hvilke målgrupper/interessenter som kan påvirke dette målet, og målgruppene knyttes til effektmålet. For hver målgruppe utarbeides det bruksmål som sier noe om hvilke behov og forventninger målgruppene har. Et bruksmål kan være knyttet til flere målgrupper. I dette eksemplet kan en *saksbehandler* være en målgruppe, og et bruksmål kan være at saksbehandlere *vil ha færre ufullstendige søknader.*

Neste trinn i effektkartet er å definere ulike tiltak knyttet til hvert bruksmål. Tiltakene sier noe om hvordan bruksmålene kan nås. Et tiltak kan være en endring i organisasjonen, endringer i manuelle rutiner, og det kan være utformingen av et saksbehandlingssystem. Et tiltak kan inneholde flere mindre tiltak.

I effektkartet skal det også legges inn målepunkter. I mitt eksempel inneholder effektmålet allerede et målepunkt: at saksbehandlingstiden skal *reduseres med 30 prosent.* Utover dette bør det også legges inn målepunkter på bruks-

194 Se nærmere Gilb (2005)
195 Se Adzic (2012)
196 Se Ottersten (2004)
197 Se Ottersten (2004) s. 46

mål og tiltak. Målepunkter kan være både estimert tid eller kostnad for et tiltak. I figur 3.5 har jeg illustrert konseptet effektkart, basert på Otterstens og Balics modell.



Figur 3.5: Effektkart som en spesifikasjon av ønskede effekter. (Oversatt fra Ottersten (2004) s. 48)

I et prosjekt med smidig programvareutvikling mener jeg effektkartet må justeres og utvides slik at det gir nødvendig forutsigbarhet med tanke på hva leverandøren skal levere. For det første vil det være hensiktsmessig å benytte begreper knyttet til brukerhistorier, se avsnitt 2.4.1.1. For det andre er det nødvendig å fremheve målepunktene – både for måling av effekt og verdi, men også når det gjelder tids- og kostnadsestimater. På denne måten kan måleparameterne ligge til grunn for prioritering og styring underveis i prosjektgjennomføringen.

Når effektkartet skal spesifisere kontraktsgjenstanden, må det inneholde alle effektmålene fra gevinstrealiseringsplanen, slik at alle interessenter har et *felles målbilde*. Det er ikke nødvendigvis slik at alle effektmålene kan realiseres i prosjektet. Det er noe kunden må ta stilling til når effektkartet skal utarbeides. Neste element i effektkartet er målgruppene, og det vil være hensiktsmessig å kun ta

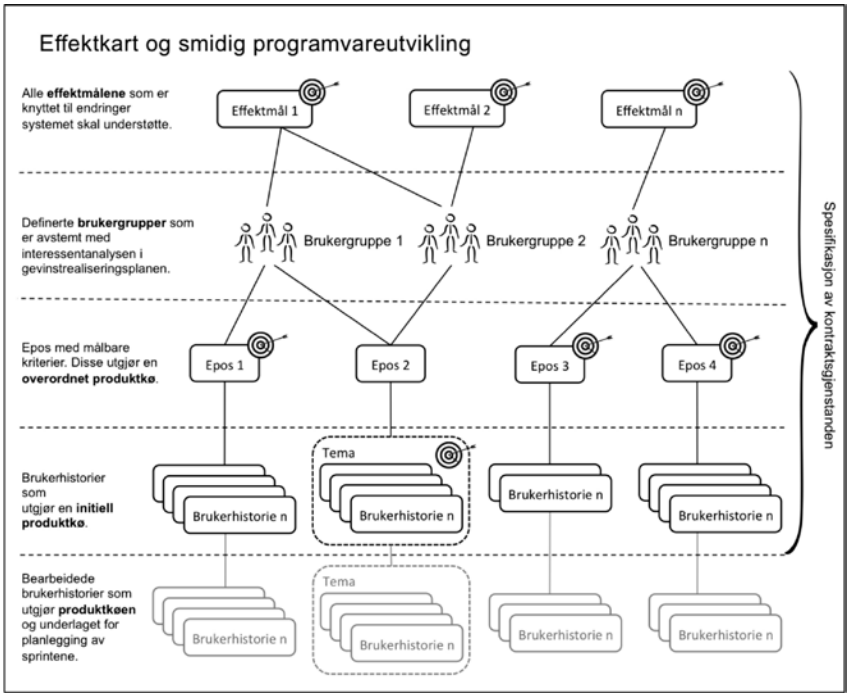
med de målgruppene som har relasjon til systemet som skal utvikles. Disse målgruppene kalles *brukergrupper*. Til hver brukergrupper utarbeides det en eller flere *epos*, og alle epos danner en *overordnet produktkø*.

Effektkartet må suppleres med *brukerhistorier,* som er knyttet til epos. Én brukerhistorie kan også bestå av flere brukerhistorier, avhengig av ønsket detaljeringsgrad. Til slutt omgjøres hver brukerhistorie til en eller flere oppgaver. Som nevnt gjøres detaljering og oppgavedefinisjon ved sprintplanleggingen. Hvor detaljert effektkartet skal være, varierer med tanke på hvilke type prosjekt, størrelse på prosjektet, graden av usikkerhet og risiko, samt hvilken prismodell som skal benyttes.[198] Som nevnt kan et system spesifiseres meget detaljert og for så vidt uttømmende ved bruk av epos, brukerhistorier og oppgaver. Da vil man i praksis ha en kravspesifikasjon på samme måte som i et plandrevet prosjekt. Imidlertid er forskjellen at kravene har en tydelig kobling til effektmålene, på liknende vis som Dragsted tar til orde for med såkalt *mapping*.[199] I et prosjekt som benytter en smidig utviklingsmetode vil det ikke være hensiktsmessig å legge til grunn et slikt uttømmende effektkart, fordi fleksibiliteten vil bli betydelig redusert. Et utgangspunkt kan være å legge til grunn et effektkart som inneholder epos og brukerhistorier på et nivå som ikke legger føringer på *hvordan* funksjonaliteten skal utvikles.

Det er også viktig å legge inn målepunkter i effektkartet – både knyttet til effektmålet men også til epos eller grupper av brukerhistorier (tema). Dette kan for eksempel være tall som sier noe om en relativ verdi for egnethet til å nå effektmålet. Det kan også være mer konkrete tall, for eksempel med forventet endring i tidsbruk for en av brukernes arbeidsoppgaver ved bruk av systemet. I figur 3.6 illustrerer jeg konseptet med effektkart for spesifikasjon av kontraktsgjenstanden.

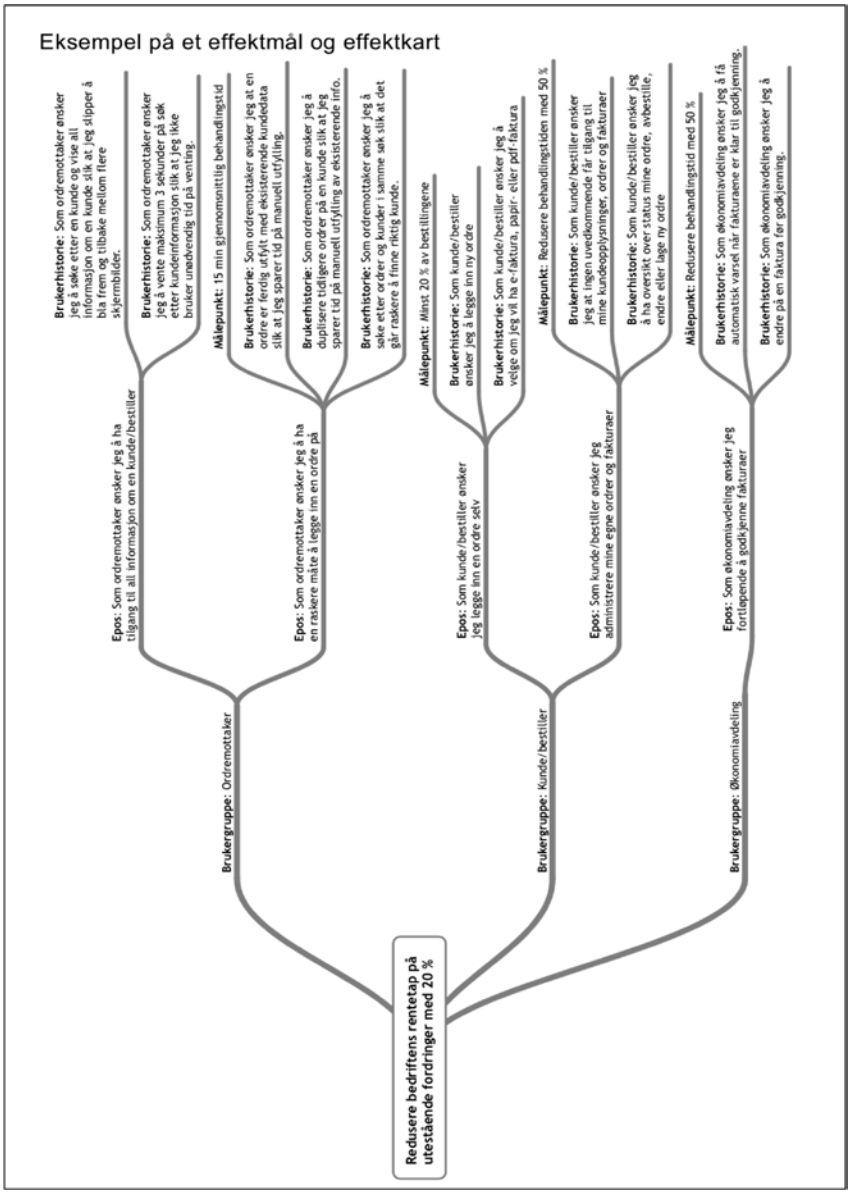
198 Se nærmere om prismodeller i underkapittel 3.6.2

199 Se underkapittel 3.5.2



Figur 3.6: Effektkart som spesifikasjon av kontraktsgjenstanden

I figuren over viser jeg at elementene som skal ligge til grunn ved kontraktsinngåelsen er *effektmål, brukergrupper, overordnet produktkø* og *initiell produktkø*. Jeg viser også at selve produktkøen først eksisterer etter at kontrakten er inngått, og at denne er en bearbeidet versjon av den initielle produktkøen. Likevel er det nødvendig at elementene i produktkøen er relatert til den initielle produktkøen, slik at koblingen til effektmålene eksisterer underveis i prosjektgjennomføringen. Koblingen gjør også at det blir tydeligere hvilke endringer som krever endringshåndtering og ikke. Jeg kommer nærmere tilbake til endringshåndtering i underkapittel 3.6.4. I figur 3.7 har jeg laget et eksempel på hvordan et effektkart med et effektmål kan se ut, og i vedlegg 5.3 finnes det samme effektkartet i skriftlig form.



Figur 3.7: Eksempel på et effektmål med tilhørende effektkart

3.6 Andre elementer i en smidig kontrakt

Når en endelig og uttømmende kravspesifikasjon ikke ligger til grunn, er det nødvendig å regulere selve utviklingsprosessen i kontrakten.[200] Dette kan gjøres ved å beskrive den smidige metoden som skal benyttes.[201] I dette kapittelet vil jeg skissere hvordan prosessen og oppgavefordelingen kan reguleres i kontrakten. Vederlaget en sentral del av alle kontrakter, og jeg beskriver noen prismodeller jeg mener kan være aktuelle i en smidig kontrakt. Videre foreslår jeg en alternativ mekanisme for konfliktløsning. Til slutt ser jeg på forholdet til prosjektstyring og endringshåndtering.

3.6.1 Utviklingsprosessen

Jeg tar utgangspunkt i rammeverket Scrum når jeg skal ta for meg utviklingsprosessen, slik den er beskrevet i underkapittel 2.4.1. Utviklingsprosessen består av flere sprinter (iterasjoner). Selv om resultatet etter hver sprint er et inkrement som er *ferdig*, så vil det i praksis *ikke* være slik at hvert inkrement blir en leveranse som skal installeres og settes i produksjon – en såkalt *release* – som jeg i den videre fremstillingen omtaler som en *delleveranse*. Ofte vil flere inkrementer ha gjensidige avhengigheter som gjør at disse må tilhøre samme delleveranse. Derfor vil størrelsen på en delleveranse variere underveis i utviklingsprosessen. Når en delleveranse skal settes i produksjon, er det nødvendig med overleveringsprosedyrer, se avsnitt 3.6.1.1.

Ansvar for produktkøen

Produkteier (kunden) har ansvaret for produktkøen, og må starte arbeidet med denne umiddelbart etter kontraktsinngåelsen. Ansvaret for produktkøen innebærer å jobbe videre med effektkartet, og spesifisere og detaljere brukerhistorier (funksjonelle og ikke-funksjonelle krav), registrere feil som skal rettes og ønskede endringer i allerede utviklede funksjoner. Produkteier skal sørge for at produktkøen er åpen og tilgjengelig for alle interessenter. Produkteier bearbeider produktkøen fortløpende uavhengig av sprintene. Dette innebærer også at elementene i produktkøen skal prioriteres basert på en vurdering av:

- hvor egnet elementet er til å nå effektmålet
- elementets utviklingsomfang sammenliknet med eventuelle andre elementer som også er egnet til å nå det samme effektmålet

Scrummasteren og utviklingsteamet (leverandøren) er forpliktet til å bistå produkteieren i det løpende arbeidet med produktkøen. Dette innebærer at de skal

200 Jf. Føyen (2006) s. 130

201 Jf. Udsen (2013b) s. 6

gjøre produkteieren i stand til å beskrive elementene i produktkøen på en slik måte at alle teammedlemmene har nok informasjon til å vite hva som skal utvikles. I tillegg skal utviklingsteamet estimere elementene i produktkøen. Utviklingsteamet skal ikke bruke mer enn en viss andel av en sprint til dette arbeidet, for eksempel ti prosent.

Krav til et inkrement
Produktet etter utviklingsarbeidet i en sprint skal være et inkrement som er en fungerende del av den totale løsningen, og inkrementet skal defineres som *ferdig*. Dette er et viktig kriterium ved smidig programvareutvikling, og er knyttet til manifestets punkt «programvare som virker fremfor omfattende dokumentasjon». Erfaringsmessig er det en vanskelig oppgave for utviklingsteamet å etablere en arbeidsmåte som gjør at inkrementene tilfredsstiller et slikt kriterium.[202] Før prosjektets første sprint skal gjennomføres må det etableres en definisjon av hva som menes med at et inkrement er *ferdig*. Det er scrumteamet (kunde og leverandør) som er ansvarlige for å utarbeide hvilke kriterier som skal legges til grunn for at et inkrement er *ferdig*. Kriteriene angir et kvalitetsnivå, og gjelder for alle inkrementer. Kriteriene kan endres underveis, men skal godkjennes av begge parter. Eksempler på slike kriterier er at:

- inkrementet skal være testet av utviklerteamet
- inkrementet skal ha en minimumsdekning på enhetstester
- enhetstestene skal være gjennomført uten feil
- enhetstester for tidligere inkrementer skal være gjennomført uten feil (regresjon)
- avtalt kodestandard skal være brukt
- inkrementet skal være installert i kundens testmiljø
- inkrementet skal oppfylle akseptansekriteriene i produktbackloggen
- inkrementet ikke har flere feil enn avtalt

Det skal alltid være definert tilstrekkelige kriterier slik at et inkrement skal kunne installeres og settes produksjon. Produkteier er ansvarlig for at de til enhver tid gjeldende kriteriene er tilgjengelig for alle interessenter i prosjektet.

Gjennomføring av en sprint
Alle sprintene skal ha samme lengde som skal være fast under kontraktsperioden, for eksempel fire uker. Det er en forutsetning for å starte en sprint (iterasjon) at produktkøen har tilstrekkelige elementer til utviklingsarbeid for minst to sprinter. Kravene til et inkrement må være godkjent før en sprint igangsettes. En ny sprint starter straks etter at forrige sprint er avsluttet. En sprint skal, som

202 Jf. Cohn (2010) s. 258

vist i underkapittel 2.4.1, inneholde følgende møter og aktiviteter: sprintplanlegging, utviklingsarbeid, daglig scrummøte, sprintreview og sprint retrospektive. Hvis uforutsette forhold gjør at en sprint ikke kan gjennomføres i henhold til sprintbackloggen, er scrummasteren forpliktet til å underrette produkteier. Produkteieren er ansvarlig for å sette i gang tiltak.

Sprintplanlegging
Før utviklingsarbeidet starter skal scrumteamet (kunde og leverandør) holde et sprintplanleggingsmøte. Møtet skal ha en varighet på maksimum åtte timer. Scrummasteren er ansvarlig for at møtet finner sted og holdes innenfor tidsrammen. Hensikten med møtet er å utarbeide en sprintbacklogg som er grunnlaget for arbeidet som skal gjøres i en sprint. Scrumteamet samarbeider om hvilke elementer fra den prioriterte produktkøen som skal tas med i sprintbackloggen. Produkteier er ansvarlig for å utarbeide akseptansekriterier knyttet til hvert element. Scrumteamet skal også vurdere og dokumentere hvilke risikoer som kan hindre fremdriften. Både produkteier og scrummaster skal godkjenne sprintbackloggen før utviklingsarbeidet kan starte.

Utviklingsarbeidet
Utviklingsteamet starter arbeidet med utviklingen av sprintens inkrement når sprintbackloggen er godkjent. Produkteier er forpliktet til å bistå utviklingsteamet med avklaringer. Utviklingsarbeidet består i nødvendige aktiviteter for å lage et inkrement som er *ferdig*. Utviklingsteamet skal oppdatere sprintbackloggen med tilleggsinformasjon om oppgaver knyttet til elementene, samt informasjon om hvilke oppgaver som er løst. Det har også ansvar for at produktbackloggen er åpen og tilgjengelig for alle interessenter.

Daglig scrummøte
Scrummasteren er ansvarlig for at det holdes daglige scrummøter til faste tider. Et scrummøte skal ha en varighet på 15-30 minutter, og utviklingsteamet har plikt til å møte. Produkteieren har rett til å møte. Hvert medlem av utviklingsteamet skal redegjøre for:

- hva som er gjort siden forrige scrummøte
- hva skal gjøres før neste scrummøte
- om det er forhold som kan hindre oppnåelsen av sprintmålet

Sprintreview
Et sprintreviewmøte skal avholdes før sprinten avsluttes, og skal ha en varighet på maksimum fire timer. Scrummasteren er ansvarlig for at møtet finner sted, og scrumteamet er forpliktet til å møte. Produkteier er ansvarlig for å innkalle

øvrige interessenter som skal delta. Utviklingsteamet skal presentere inkrementet, og redegjøre for hvordan utviklingsarbeidet er gjennomført. Scrumteamet skal sammen vurdere om inkrementet oppfyller de fastsatte kriteriene, i tillegg til akseptansekriteriene i produktbackloggen. Produkteieren godkjenner eller underkjenner elementene i henhold til kriteriene. Elementer som oppfyller kriteriene kan ikke underkjennes. Elementer som underkjennes skal tilbake til produktkøen. Elementer som godkjennes skal dokumenteres og tas ut av produktkøen. Produkteieren er ansvarlig for videre håndtering av inkrementet med godkjente elementer. Videre håndtering vil typisk være samarbeid med brukere om videre testing, samt rapporteringer til prosjektorganisasjonen. Møtet kan også brukes til å legge inn nye elementer i produktkøen, samt vurdering av elementenes prioritering.

Sprint retrospective
Scrumteamet skal holde et sprint retrospectivemøte som avslutning på en sprint. Scrummasteren er ansvarlig for at møtet finner sted, og varigheten er maksimum tre timer. Hensikten med møtet er et tilbakeblikk på sprinten for å vurdere hvordan utviklingsarbeidet er gjennomført. Vesentlige forhold som var vellykkede og forhold som kan forbedres skal dokumenteres. Det skal utarbeides en plan for hvordan forbedringene skal gjennomføres. Denne planen skal være med i grunnlaget for planleggingen av neste sprint.

3.6.1.1 Delleveransen
En delleveranse består av et eller flere inkrementer. Produkteier er ansvarlig for å utarbeide en testplan for en delleveransen. Testplanen skal inneholde de enkelte inkrementenes krav, samt elementenes akseptansekriterier. Produkteier er ansvarlig for at delleveransen testes innenfor testperioden. Testperiodens lengde er avhengig av størrelsen på delleveransen. Elementer som ikke oppfyller testkriteriene skal dokumenteres, og leverandøren er ansvarlig for å rette feilene. Når testperioden er avsluttet skal produkteier godkjenne eller underkjenne delleveransen. Godkjente delleveranser skal ansees som levert, og kan settes i produksjon.

3.6.2 Prismodell
Betalingen er en av kundens viktigste plikter i en tilvirkningskontrakt. Vederlaget baserer seg på prinsippet om ytelse mot ytelse. I komplekse tilvirkningsavtaler kan beregning og fastsettelse av vederlaget være komplisert.[203] Denne utfordringen øker i prosjekter med smidig programvareutvikling når det som skal utvikles ikke er endelig fastsatt ved kontraktsinngåelsen. I tillegg legger prosjektgjennomføringen opp til en annen risikofordeling enn de tradisjonelle kon-

203 Jf. Haaskjold (2013) s. 579

traktene, og prismodellen kan får betydning for denne risikofordelingen.[204] Det er vanlig å skille mellom to utgangspunkter for vederlagsberegning: *fast pris* og *regningsarbeid*. Fast pris blir ofte benyttet i kontrakter hvor kunden ønsker høy forutsigbarhet med tanke på både kostnader, leveringstid og omfang. Regningsarbeid blir typisk brukt i prosjekter hvor kontraktsgjenstanden ikke har en uttømmende spesifikasjon, og leverandøren har en innsatsforpliktelse, se figur 1.1.

I en fastpriskontrakt, hvor leverandøren har en resultatforpliktelse med tanke på omfang og leveringstid, har leverandøren risikoen for utforutsette merutgifter eller høyere tidsforbruk. Men leverandøren kan også oppnå gevinst om tilvirkningen viser seg å bli lettere enn beregnet. Jeg har tidligere vært inne på at it-prosjektene gjerne koster mer og tar lenger tid en planlagt, snarere enn det motsatte. I fastpriskontrakter vil kundens endringsbehov føre til tilleggsvederlag for leverandøren.[205] Usikkerheten knyttet til it-prosjekter vil føre til en rekke endringer underveis i prosjektgjennomføringen.[206] Dette gjør at systemet blir dyrere enn det kunden har lagt til grunn ved kontraktinngåelsen. Et annet aspekt er hvis kunden velger en leverandør med lav fast pris i en tilbudsprosess. Da kan det bli et problem hvis leverandøren ønsker å hente inn fortjeneste via endringsordrer som gir tilleggsvederlag.[207] Når leverandøren bærer en stor del av vederlagsrisikoen, og dermed er eksponert for tap i prosjekter med stor usikkerhet, vil han aktivt sikre egne interesser. Dette kan skape konflikter om hva som faktisk skal utvikles og hvilke endringer som kan utløse tilleggsvederlag. Leverandørene vil derfor vegre seg mot å inngå fastpriskontrakter i slike prosjekter.[208] I et prosjekt som benytter en smidig utviklingsmetode, vil usikkerheten med hva som skal leveres være høy, og en fastpriskontrakt vil derfor være lite egnet. Dragsteds og Klints undersøkelse underbygger dette synet, fordi det kommer klart frem at fast pris ikke er en foretrukket prismodell ved bruk av smidige metoder.[209]

Det andre utgangspunktet for vederlagsberegning er regningsarbeid, som på engelsk kalles «time and material» (T&M). Ved bruk av en slik beregning skal kunden betale for kostnadene leverandøren har pådratt seg, og som regel med et påslag til dekning av risiko og fortjeneste. Dette gjelder uavhengig av om kostnadene er resultat av merutgifter på grunn av utforutsette vanskeligheter. En slik prismodell gjør at det er kunden som i stor grad bærer vederlagsrisikoen.[210] Dette gjelder likevel ikke ubetinget. I Rt. 1969 s. 1122 konkluderte Høyesterett med at kunden ikke skal betale for merkostnader som skyldes uforsvarlige for-

204 ibid. s. 581
205 ibid. s. 584
206 Se nærmere underkapittel 2.5.3
207 Se Poppendieck (2003) s. 167
208 Jf. Føyen (2006) s. 447
209 Se Dragsted (2013) s. 21
210 Jf. Føyen (2006) s. 446

hold som leverandøren har ansvaret for.[211] En slik begrensning er også regulert i NS 8407 pkt. 30.1: «Regningsarbeider skal drives rasjonelt og forsvarlig».

Ofte blir regningsarbeid relatert til bistandskontrakter hvor leverandøren har en innsatsforpliktelse. I et slikt prosjekt er det kunden som har risikoen for prosjektgjennomføringen og resultatet. Utfordringene oppstår i kontrakter hvor det legges opp til en risikofordelingen mellom kunde og leverandør når det gjelder leveranseansvaret. I en smidig kontrakt vil vederlagsberegning basert på regningsarbeid være et naturlig utgangspunkt. Kunden får et større ansvar for gjennomføringen av prosjektet, blant annet ved at produkteier prioriterer produktkøen og godkjenner sprintbackloggen. Et slikt ansvar for kunden vil være vanskelig å kombinere med for eksempel en fast pris på det som skal utvikles. Hvis regningsarbeid som prismodell skal benyttes, er det en forutsetning at partene har en viss grad av tillit og lojalitet til hverandre. Når denne prismodellen benyttes har leverandøren en plikt til lojalt å sørge for at kundens interesser blir ivaretatt, og ikke utføre arbeider som er mer kostbare enn nødvendig.[212]

Det hevdes at ved mangel på tillit og lojalitet kan leverandøren være tjent med å trekke ut tiden, fordi fortjenesten er avhengig av antall leverte timer.[213] I en norsk undersøkelse kom det frem at utvikling av programvare basert på regningsarbeid hadde de største kostnadsoverskridelsen, sammenliknet med blant annet fastpriskontrakter.[214] Årsaken til dette var nettopp at leverandøren ble motivert til å utvikle unødvendig funksjonalitet, og såkalt *gold plating*.[215] Slike forhold kan medføre at kunden etablerer relativt omfattende kontrollmekanismer med tanke på medgått tid, og transaksjonskostnadene blir høye. Imidlertid kan kommunikasjon, åpenhet og hyppige leveranser av inkrementer i Scrum, redusere behovet for ytterligere kontrollmekanismer.[216] I tillegg vil bruken av effektkart som styringsmekanisme, kontinuerlig sikre at det som utvikles er egnet til å nå effektmålene.[217] Etter mitt syn vil en vederlagsberegning basert på regningsarbeid være en riktig løsning i en smidig kontrakt.

Ved fravær av tillit mellom partene, eller hvis man ønsker å stimulere til effektivitet og kvalitet, er det ikke uvanlig at avtalt prismodell legger til rette for økonomiske insentiver. Disse insentivene kan være av både positiv og negativ art. Negative insentiver regnes som sanksjoner, og eksempler på slike kan være dagbøter og redusert timepris. Positive insentiver er at leverandøren får en økonomisk påskjønnelse etter nærmere kriterier for effektivitet og kvalitet. Slike

211 Se Rt. 1969 s. 1122 (s. 1124)

212 Jf. Giverholt (2012) s. 449

213 Jf. Poppendieck (2003) s. 167

214 Se Moløkken-Østvold (2007) s. 79

215 *Gold plating* er en betegnelse som brukes på unødvendig forbedring av utviklede funksjoner som ikke gir verdi for kunden.

216 Se Poppendieck (2003) s. 168

217 Jf. Ottersten (2004) s. 23

påskjønnelser kan for eksempel være bonus og økt timepris. Ved bruk av smidige metoder vil kundens medvirkning og risikofordelingen mellom partene ha stor betydning for gjennomføringen av prosjektet. Dette vil kunne påvirke hvordan insentivene slår ut for leverandøren, og det oppstår et spenningsforhold mellom forventet og faktisk mulighet for insentiver.[218] I tillegg viser forskning at slike insentiver kan føre til spekulativ adferd hos partene, redusert åpenhet og lavere kvalitet.[219] Ved for eksempel bruk av bonus for å ferdigstille prosjektet tidligere enn planlagt, kan det føre til at leverandøren utvikler programvare med dårligere kvalitet for å øke utviklingshastigheten. På en annen side vil bonus basert på få feil i leveransene kunne øke kvaliteten. De ulike modellene for insentiver kan etter min mening bli for kompliserte slik at de mister mye av sin praktiske betydning. Larman foreslår tre alternative løsninger til slike insentiver:[220]

- ikke bruke insentiver – beholde enkelhet i prismodellen
- at kunden kan avslutte kontraktsforholdet etter hver iterasjon
- at partene fordeler de økonomiske konsekvenser av over- eller underforbruk

Å ikke bruke insentiver, men å gjennomføre prosjektet med en enkel prismodell, vil i praksis være bruk av vederlagsberegning basert på regningsarbeid. At kunden kan av slutte arbeidet underveis i prosjektgjennomføringen kan løse flere typer av konflikter som kan oppstå – inkludert uenighet om pris, fremdrift og kvalitet. Jeg kommer nærmere tilbake til disse situasjonene i underkapittel 3.6.5. En vanlig prismodell for å fordele de økonomiske konsekvensene ved over- eller underforbruk er ved bruk av såkalt *målpris*, som på engelsk kalles «target cost». En kontrakt med målpris er en mellomting mellom fast pris og regningsarbeid. Målprisen består som regel av to eller flere komponenter som beregnes i en tidlig fase av prosjektet. Først utarbeides det et estimat på utviklingstid. I tillegg kommer påslag i prosent for usikkerhet basert på en risikoanalyse eller kun et gevinstpåslag i prosent, for eksempel 15 prosent. Utgangspunktet er at faktisk timeforbruk knyttet til leveransene blir avstemt med målprisen. Hvis faktisk kostnad overstiger målprisen, fordeles overstigende kostnad på begge parter. Tilsvarende hvis faktisk kostnad er lavere enn målprisen, fordeler partene gevinsten. Det finnes en rekke varianter for å beregne målpris og avvik fra denne. I tabell 3.1 viser jeg et eksempel på bruk av målpris basert på at kunden og leverandøren fordeler over- og underforbruk med 50 prosent.

218 Jf. Føyen (2006) s. 452-453
219 Se Larman (2010) s. 514
220 l.c.

Tabell 3.1: Eksempler på bruk av målpris ved over- og underforbruk. (Basert på Larman (2010) s. 541)

Estimert totalpris	Påslag 15 prosent	Målprisen	Faktiske kostnader	Justering med 50 prosent	Kundens betaling	Fortjeneste for leverandør
1 000 000	150 000	1 150 000	1 100 000	+ 50 000	1 200 000	100 000
1 000 000	150 000	1 150 000	900 000	- 50 000	1 100 000	200 000

I tabellens første rad vises et tilfelle hvor faktiske kostnader overstiger estimerte kostnader, og kunden må betale halvparten av de økte kostnadene, mens leverandørens fortjeneste reduseres tilsvarende. I andre rad er den faktiske kostnaden lavere enn estimert; kunden betaler mindre, mens leverandøren øker fortjenesten. Hvis målprisen skal justeres underveis i prosjektet blir dette håndtert som en endring av kontrakten. Forskning viser at bruk av målpris i kontrakter kan redusere risikoen for kostnadsoverskridelser.[221] Bruk av målpris motiverer også til bedre samarbeid og prioritering av utvikling som gir ønskede effekter.[222] Derfor vil denne prismodellen i utgangspunktet være egnet for en smidig kontrakt. I PS2000-kontraktene er det en målprismodell som ligger til grunn for vederlagsberegningen.

Den store ulempen med målpris er, etter min mening, nødvendigheten av et bindende estimat som grunnlag for beregningen. For at et slikt estimat skal tilrettelegge for en reell gevinstmulighet, forutsetter det at omfanget er relativt klart definert allerede i starten av prosjektet. Dette medfører at fleksibiliteten for et prosjekt som benytter en smidig metode, blir vesentlig redusert. Det kan hevdes at mulighetene for å endre målprisen underveis motvirker redusert fleksibilitet, men etter mitt syn vil prismodellen da få mer preg av en fast pris med mulighet for tilleggsvederlag.

En prismodell jeg synes er interessant, er Gilbs forslag til en såkalt *no cure, no pay*.[223] Utgangspunktet i denne modellen er at det kun er levert programvare som gir påviselig nytteeffekt for kunden som gir grunnlag for opparbeidelse av et vederlagskrav. Jeg har i underkapittel 3.4.2 vært inne på Gilbs metoder for å måle effekt og nytte av utviklet programvare. Når det gjelder en prismodell med no cure, no pay, mener han at det som utvikles skal måles mot definerte målbare kriterier, og at leverandøren kun får betalt for programvare som tilfredsstiller målekriteriene. Modellen er interessant fordi det er nettopp å nå effektmålene som gjør at gevinstene kan realiseres. Imidlertid tror jeg at leverandørene vil

221 Se Moløkken-Østvold (2007) s. 79
222 Jf. Poppendieck (2003) s. 171-172
223 Se Gilb (2006)

være tilbakeholdne med å inngå slike kontrakter, fordi leverandørene ikke har tilstrekkelig kontroll på øvrige faktorer som kan påvirke den faktiske effekten av utviklet funksjonalitet.

Selv om valg av prismodell er en sentral komponent i en smidig kontrakt, kan ikke den alene være bærende for reguleringen av risikofordelingen. Prismodellen som velges er kun en av flere mekanismer i kontrakten. Den må sees i sammenheng med andre vilkår knyttet til utviklingsprosessen, konfliktløsning og kontraktsbrudd.

3.6.3 Prosjektstyring

Prosjektstyring kan beskrives som «the art and science of converting vision into reality.»[224] Jeg oppfatter denne beskrivelsen som at det handler om å gjennomføre et vellykket prosjekt. Et vellykket prosjekt måles på i hvilken grad visse *suksesskriterier* er oppfylt, se underkapittel 1.4.3. Hvilke suksesskriterier som ligger til grunn vil variere fra prosjekt til prosjekt, men som regel har et prosjekt rammer for budsjett og tid. I noen prosjekter er det kritisk at hele eller deler av et system kan tas i bruk innen en gitt dato. I andre prosjekter er kanskje budsjettrammen absolutt, mens tidsrammen er mindre kritisk. Øvrige suksesskriterier er knyttet til kvalitet (prosjektresultat) og effektmål. I manifestet for smidig programvareutvikling er det første punktet som følger:

> «Personer og samspill fremfor prosesser og verktøy»

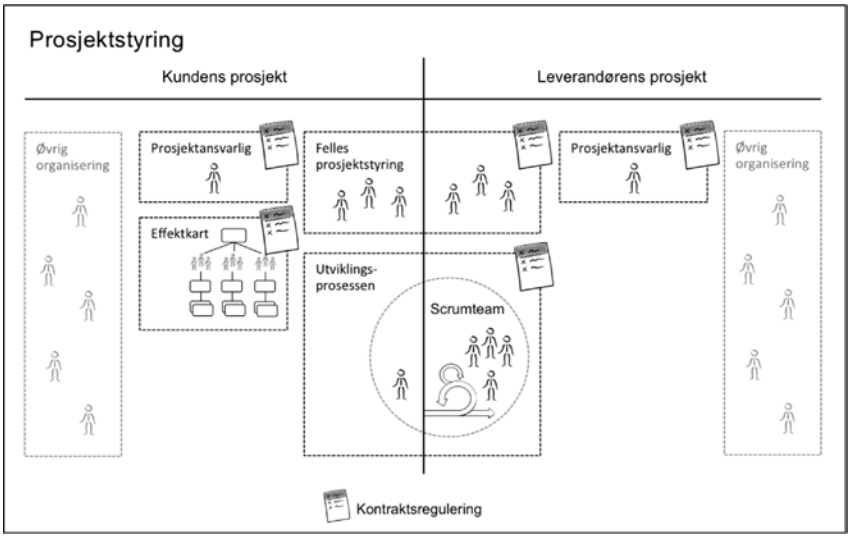
Dette punktet innebærer at blant annet scrumteamet skal ta ansvar for at effektmålene nås, og at teamet er selvorganiserende slik at optimale løsninger oppnås på en mest mulig effektiv måte. I tillegg er det viktig med god og hyppig kommunikasjon både internt i prosjektet og eksternt med tanke på interessenter. Disse prinsippene er opphav til oppfatninger om at det ikke er behov for prosjektstyring ved bruk av smidige utviklingsmetoder. Slike oppfatninger henger trolig sammen med at innholdet i den tradisjonelle rollen som prosjektleder er endret i prosjekter som benytter en smidig metode.

I et plandrevent prosjekt vil prosjektlederens oppgaver være alt fra å sikre målstyring i henhold til effektmål og gevinstrealisering, til detaljert oppgave- og ansvarsfordeling med tilhørende kontroll og evaluering av utførelsen. I tillegg skal prosjektlederen lage planer i henhold til tids- og kostnadsrammene og kvalitetsmål, sikre gode arbeidsforhold og støttefunksjoner for prosjektmedlemmene, samt sørge for nødvendig kommunikasjon internt og eksternt. Ved bruk av Scrum er mange av de tradisjonelle oppgavene til prosjektlederen overført til scrumteamet.[225] Scrum er tydelig på roller og ansvar, samt hvordan gjennom-

224 Se Turner (1996) s. 6

225 Se Cohn (2010) s. 139-141

føringen skal skje for å nå effektmålene. Derfor blir Scrum også omtalt som en *prosjektstyringsmetodikk*.[226] Det selvorganiserte scrumteamet har ansvaret for individuell oppgavefordeling, identifisering av risiko, sikre kvalitet i hvert inkrement, utvikle mot prioriterte effektmål, estimering etc., se underkapittel 2.4.1. Selv om Scrum benyttes som prosjektstyringsmetodikk, er det nødvendig med roller som kan ivareta funksjoner som ikke ligger innenfor scrumteamets ansvarsområder. Et utviklingsprosjekt, slik det er definert i denne oppgaven, innebærer at kunden oppretter et prosjekt hvor en leverandør skal levere en løsning i henhold til et effektkart ved bruk av Scrum. Både kunden og leverandøren vil typisk opprette sine interne prosjekter. Prosjektstyring med tanke på kontraktsregulering handler om hvordan det felles prosjektet skal reguleres. I figur 3.8 illustrerer jeg skillet mellom partenes interne prosjekter, og hvilke deler som må reguleres i kontrakten.



Figur 3.8: Styring av partenes felles prosjekt, med hvilke deler som må reguleres i kontrakt.

Kundens prosjektansvarlig skal sørge for at effektkartet utarbeides som grunnlag for kontraktsinngåelsen. Det er viktig at det er bred enighet om effektkartets utforming i kundens organisasjon. Den prosjektansvarlige må også bruke kundens prosjektmandat som grunnlag for prosjektgjennomføringen. I dette mandatet vil det som oftest ligge rammer for budsjett og leveringstid. Tilsvarende vil

226 Jf. Karlsen (2013) s. 261

leverandørens prosjektansvarlig sørge for at tilstrekkelige ressurser og kompetanse kan leveres i henhold til inngått kontrakt. Partenes respektive prosjektansvarlige er roller som må reguleres i kontrakten, og vil være de som opptrer på vegne av partene. I SSA-S er dette regulert i pkt. 1.5, og i PS2000 er tilsvarende bestemmelser i pkt. 2.1.2. De prosjektansvarlige vil også følge opp andre områder i partenes interne prosjekt, men disse oppgavene er ikke den del av kontrakten.

I et prosjekt som benytter smidige utviklingsmetoder er det nødvendig med tett samarbeid mellom partene. Dette reguleres i Scrum og utviklingsprosessen. I tillegg er det nødvendig at kontrakten regulerer et felles samarbeids- eller styringsorgan som er ansvarlige for andre områder enn de som dekkes av scrumteamet. Dette er områder som overordnet planlegging av fremdrift og leveranser, risikostyring, kommunikasjon og styring i henhold til effektkart (effektstyring), konfliktløsning, endringshåndtering og øvrig kontraktsoppfølging. Dette kaller jeg *felles prosjektstyring*. I PS2000 Smidig skal det opprettes en *koordineringsgruppe*, jf. bilag B.2.1. Kontrakten må regulere hvilke roller og personer som skal utgjøre gruppen som er ansvarlig for felles prosjektstyring. Typisk vil gruppen inneholde partenes prosjektansvarlige, produkteier(e), scrummaster(e) og eventuelt øvrige interessenter hos partene.

Planlegging

I følge Cohn kan planlegging sees på som «[…] a quest for value.»[227] Det vil si at planlegging må til for at prosjektet skal nå effektmålene. Planlegging reduserer risiko og usikkerhet, skaper bedre beslutningsgrunnlag og gir grunnlag for tillit og utveksling av informasjon.[228] I tillegg har prosjektene som regel budsjett- og tidsrammer som er viktige styringselementer i planleggingen. Basert på effektkartet og et estimat på epos og brukerhistorier, skal partene i fellesskap utarbeide en overordnet plan for prosjektgjennomføringen. Det vil si å estimere antall sprinter samt når det skal være delleveranser. Planen skal være dynamisk, og skal endres underveis når estimatene blir mer presise. I tillegg vil identifiserte risikoer og erfaring som opparbeides underveis kunne medføre endringer i planen. I planleggingen må effektkartet være styrende for delleveranser og hvilke målbare effekter som er forventet i hver delleveranse.

Effektstyring

En annen viktig oppgave i den felles prosjektstyringen er såkalt *effektstyring*. Dette er nødvendig for underveis å kunne prioritere, og å styre etter effektmålene i effektkartet. Ottersten og Balic anbefaler å ha en egen effektstyringsgrup-

227 Se Cohn (2005) s. 5
228 l.c.

pe,[229] men etter mitt syn er dette en oppgave som vil egne seg i gruppen for felles prosjektstyring. Men, det er viktig at kundens prosjektansvarlig, i samarbeid produkteier, sørger for å kommunisere og beslutte prioriteringer i henhold til effektkartet med øvrige interessenter. Disse beslutningen tar produkteier tilbake til scrumteamet.

Oppfølging
Det er vanlig å ha ulike kontroll- og evalueringsmekanismer i et prosjekt, men behovet for eksplisitte mekanismer ved bruk av smidige metoder er redusert. Det er fordi de hyppige leveransene av inkrementer gjør at utfordringer og risikoer fortløpende blir avdekket ved empirisk kontroll. Spesielt viktig er evalueringen av inkrementenes bidrag til å nå effektmålene innenfor planlagt tids- og kostnadsrammer. Derfor er det nødvendig at den felles prosjektstyringen praktiserer et såkalt *åpen bok-prinsipp*, basert på at partene har fullt innsyn i prosjektets økonomi. Dette er et sentralt aspekt i såkalte samspillkontrakter.[230] Spesielt viktig er dette hvis prismodellen har insentivelementer. I en smidig kontrakt er det nødvendig at begge parter er forpliktet til lojalt å varsle den andre parten om forhold som kan påvirke prosjektgjennomføringen. Selv om bakgrunnsretten inneholder et generelt prinsipp om lojalitetsplikt i kontraktsforhold, mener jeg at det i en smidig kontrakt bør tas inn en tydelig bestemmelse om den gjensidige lojalitetsplikten. Dette er også anbefalt for kontrakter som legger til rette for tett samarbeid mellom partene.[231]

3.6.4 Endringer

Ved bruk av Scrum som utviklingsmetode, basert på effektkart som spesifikasjon av kontraktsgjenstanden, åpnes det for høy grad av fleksibilitet. Dette gjør at endringshåndteringen underveis i prosjektet ikke får samme betydning som det tradisjonelle utgangspunktet med uttømmende spesifikasjoner ved kontraktsinngåelsen. Effektkartet beskriver ikke hvilke funksjoner som skal utvikles og hvordan de skal fungere. Dette avklares underveis i gjennomføringen. Det betyr at scrumteamet har stor valgfrihet med tanke på utviklingen, og valgene som tas er ikke knyttet til endringer i den opprinnelige kontrakten, men som detaljspesifisering av effektkartet. Likevel er det behov for en viss regulering av noen typer endringer i en smidig kontrakt. Endringshåndtering er alltid knyttet til avvik fra innhold i kontrakten. At det oppstår slike avvik i en smidig kontrakt er neppe en uvanlig situasjon. I en smidig kontrakt er det mange forhold som reguleres, og endringer kan typisk være knyttet til:

229 Se Ottersten (2004) s. 60-61
230 Se Entreprenørforeningen – Bygg og Anlegg (2013) s. 11
231 Se Føyen (2006) s. 251

- justering av elementer i effektkartet
- skifte av personer i ulike roller
- endringer i budsjett og tidsrammer
- andre endringer i organisering og gjennomføring

For det første bør endringer i kontrakten gjøres med en viss formalisme med tanke på form og fremgangsmåte.[232] Det er nødvendig fordi kontrakten er inngått skriftlig, og en endring vil erstatte eller justere innholdet i kontrakten. En uryddig endringshåndtering kan lett føre til uklarheter og konflikter.[233] Likevel bør endringsbestemmelsene være enkle å bruke, slik at de blir effektive og praktiske.[234] På bakgrunn av dette er derfor nødvendig å regulere endringshåndtering i en smidig kontrakt. Endringsanmodninger kan komme fra begge parter, og disse skal drøftes i gruppen for felles prosjektstyring. Det er viktig at begge parter legger frem endringsanmodninger hvis de mener at justeringer og prioriteringer av produktkøen ikke er i tråd med effektkartet. Gruppen skal utarbeide en endringsordre som beskriver endringen og årsak til endringer, samt hvilke konsekvenser for prosjektgjennomføringen endringen fører til for fremdrift, budsjett, organisering, risiko og prismodell. Begge parter skal godkjenne endringen, som da blir en del av kontrakten for videre oppfølging. Hvis det ikke oppnås enighet om endringen får den status som omtvistet, og må håndteres ved hjelp av konfliktløsning.

Regler for preklusive frister egner seg ikke i en smidig kontrakt. Det er tillit og lojalitet som først og fremst skal være styrende for partenes plikter. Slike regler er heller ikke anbefalt i EBAs veileder for samspillentreprise.[235] Endringer er også relatert til bestemmelser om kansellering av kontrakten.[236] Hvis endringen er omfattende kan den i praksis få samme virkning som en avbestilling, se underkapittel 3.6.6.

3.6.5 Konflikthåndtering

Det optimale i prosjektgjennomføringen er å unngå konflikter. Konflikter koster tid og ressurser, og har dermed negative konsekvenser for fremdriften. Et tett samarbeid mellom partene som bygger på gjensidig tillit er et godt grunnlag for å unngå konflikter, og kontrakten skal også være utformet med tanke på konfliktforebygging. Men, som Kaasen kort oppsummerer: «Konflikter […] vil oppstå.»[237] I prosjekter med smidige utviklingsmetoder er det svært viktig at uenigheter løses effektivt underveis i gjennomføringen. Uenigheter kan fort føre til at

232 Jf. Kolrud (2009) s. 157
233 Jf. Torvund (1997) s. 123
234 Se Kolrud (2009) s. 154
235 Se Entreprenørforeningen – Bygg og Anlegg (2013) s. 32
236 Jf. Kolrud (2009) s. 160
237 Se Kaasen (2010) s. 288

konflikter forsurer samarbeidsklimaet og bygger ned etablert tillit. Konflikter bør først og fremst løses på prosjektnivå, det vil si i gruppen for felles prosjektstyring. I praksis vil en rekke uavklarte situasjoner løses ved *forhandlinger*. Men som vi har sett kan for eksempel en omtvistet endring nettopp være en konsekvens av slike forhandlinger. Da blir spørsmålet hvordan konflikter som ikke løses ved forhandlinger skal håndteres.

Det finnes ulike mekanismer for konfliktløsning, og blant mulighetene er tvisteløsning ved domstolene i henhold til tvistelovens[238] bestemmelser, eller ved voldgiftsbehandling hvis dette er avtalt. Å bringe konflikten så langt kan bli svært ressurskrevende og naturlig ødeleggende for samarbeidet. En slik løsning vil trolig være siste skanse for de fleste prosjekter. Mekling og bruk av uavhengig ekspert er andre muligheter for tvisteløsning, som er mindre ressurskrevende og mer egnet for konfliktløsning underveis i prosjektgjennomføringen.

Både SSA-S, jf. pkt. 16.3-16.4, og PS2000 Smidig, jf. pkt. 8.6.2, har et opplegg med en uavhengig ekspert og eventuelt påfølgende mekling. Eksperten skal eller bør være oppnevnt ved kontraktsinngåelsen. Ulempen ved disse ordningene er at eksperten kun får kjennskap til prosjektet når konflikten har nådd et nivå og partene har sett seg nødt til å få hjelp. En mer proaktiv tankegang som legger til rette for raskere konfliktløsning kan være såkalt *prosjektintegrert mekling*. Jeg mener det er et godt utgangspunkt for å hindre skalering, samtidig som det kan gi en mer effektiv løsning på konfliktene. Kaasen trekker frem tre elementer som utgjør kjernen i prosjektintegrert mekling:[239]

- en eller flere meklere
- trekkes inn i prosjektet fra dag én
- fører løpende kontakt med partene

Det må vurderes om mekleren skal kunne treffe bindende avgjørelse eller ikke. Kaasen påpeker at dette hovedsakelig er et teoretiske spørsmål, da partene oppfatter meklerens råd som så viktige at de i praksis vil bli fulgt opp av partene.[240] Andre momenter som må vurderes er hvor mange meklere som skal brukes, hvordan de skal jobbe i prosjektet samt omfanget av arbeidet. Her vil det i praksis være størrelsen på prosjektet som avgjør. Jo større og mer komplekse prosjektene er, jo flere meklere med ulike kompetanse og jo høyere intensitet på arbeidet kreves. I standardkontrakten NS 8407 er muligheten for prosjektintegrert mekling regulert i pkt. 50.2. Bestemmelsen er dynamisk med hensyn til utformingen av meklingen, men partene må eksplisitt avtale en slik ordning, jf. «Dersom partene er enige om det […]» i første avsnitt.

238 Tvisteloven (2005)
239 Se Kaasen (2010) s. 293
240 Jf. Kaasen (2010) s. 294-295

Min konklusjon etter denne gjennomgangen er at en smidig kontrakt må inneholde bestemmelser om at konflikter først og fremst skal løses med *forhandlinger* i gruppen for felles prosjektstyring. Konflikter som ikke lar seg løse på denne måten skal gjennomgå *prosjektintegrert mekling*. Mekleren skal oppnevnes ved kontraktsinngåelsen og delta på møter i gruppen for felles prosjektstyring. Dette gir mekleren god innsikt og et solid grunnlag for å gi råd og veiledning i en konfliktsituasjon.

3.6.6 Avbestilling og oppsigelse

Det er flere årsaker til at partene ønsker å avslutte et prosjekt før det er ferdig. I kontrakter av en viss varighet kan kundens behov endre seg slik at det ikke er formålstjenlig å fullføre prosjektet.[241] Spesielt ved bruk av smidige metoder kan det også være tilfelle at effektmålene er nådd i tilfredsstillende grad tidligere enn planlagt. Leverandøren kan også ha behov for å avslutte prosjektet hvis det ikke er mulig å gjennomføre prosjektet som planlagt.[242] I tillegg kan endringsbehov være så inngripende at det bør behandles som en avbestilling. For eksempel i NS 8407 skal endringer som overstiger en viss størrelse i henhold til kontraktsverdien regnes som avbestilling, jf. pkt. 44. Ellers kan avbestilling være et alternativ til videre skalering av en konflikt som ikke lar seg løse med forhandling eller mekling. Det er vanlig at tilvirkningskontraktene har avbestillingsadgang for kunden, og mindre vanlig at leverandøren har denne muligheten. Muligheten til avbestilling er ikke avhengig av at det konstateres kontraktsbrudd. Avbestillinger i tilvirkningskontrakter kan kun virke fremover i tid (ex nunc). En avbestillingsadgang kan også fungere som et verktøy for å balansere risikoen mellom partene.[243]

Når prosjekter benytter smidige utviklingsmetoder er det viktig å ivareta det grunnleggende prinsippet om fleksibilitet. Partene skal ha stor grad av valgfrihet, og denne friheten vil bli redusert hvis avbestillingsadgangen setter begrensninger.[244] Det tette samarbeidet mellom partene ved bruk av smidig metode gjør at også leverandøren bør ha en adgang til å avslutte samarbeidet. Det ville være uheldig om leverandøren skulle tvinges til å fortsette når forutsetningen for samarbeid ikke lenger er tilstede. Derfor anbefales det at en smidig kontrakt har bestemmelser om at begge parter har oppsigelsesadgang etter hver iterasjon/sprint.[245] Etter min mening vil en mulighet for avbestilling ved hver sprint blir for vid. Ofte vil det være slik at flere sprinter til sammen utgjør en delleveranse som skal settes i produksjon, jf. avsnitt 3.6.1.1. Derfor kan det være tapsbegrensende at avbestillinger kun kan gjøres i forbindelse med akseptansen av en del-

241 Jf. Torvund (1997) s. 130
242 Jf. Føyen (2006) s. 317
243 Jf. Udsen (2013b) s. 7
244 Se Føyen (2006) s. 328-329
245 Se Larman (2010) s. 522

leveranse og før oppstart av ny iterasjon. Da har kunden en reell mulighet til å nyttiggjøre seg av levert programvare.

En slik vid adgang til oppsigelse kan gjøre det enklere å løse fastlåste konflikter, samt begrense tap i forbindelse med kontraktsbrudd. Et annet aspekt gjelder prismodellen. Ved bruk av vederlagsberegning basert på regningsarbeid er det en forutsetning at det eksisterer en viss tillit mellom partene, jf. underkapittel 3.6.2. Hvis ikke tilstrekkelig tillit opparbeides eller forventet resultat uteblir, er en løsning på problemet å avslutte samarbeidet. En slik mulighet vil også legge et visst press på leverandøren, slik at ikke unødvendig arbeid utføres for å øke fortjenesten. Det er lagt opp til en vid oppsigelsesadgang for partene i SSA-S, jf. pkt. 2.7.1, men denne adgangen gjelder kun frem til akseptansetesting av prosjektets første delleveranse. Etter dette tidspunktet er det kun kunden som har en tradisjonell avbestillingsadgang i henhold til et økonomisk oppgjør, jf. pkt. 2.7.2.

For både kunde og leverandør vil en slik vid oppsigelsesadgang kunne få økonomiske konsekvenser. Derfor bør det utarbeides bestemmelser som regulerer et økonomisk oppgjør mellom partene ved oppsigelse. For å sikre kundens forutsigbarhet bør det stilles krav om at det skal gjennomføres prosjektintegrert mekling før leverandøren kan avbestille. Siden kunden kan benytte seg av levert programvare etter en oppsigelser er det nødvendig at kontrakten regulerer rettighetene til levert materiale. Oppsigelser skal være begrunnet, og en illojal oppsigelse kan utløse krav om erstatning.

3.7 Risikofordeling og kontraktsbrudd

Kontraktsbrudd har en side til risikofordeling og en side til sanksjoner. Sanksjonene har som formål å gjenopprette balansen i kontraktsforholdet ved kontraktsbrudd, samt gi partene insitament til riktig oppfyllelse.[246] Et av formålene med en smidig kontrakt er å bidra til vellykket gjennomføring av et prosjekt ved tett samspill mellom partene. Likevel har partene også ulike formål med prosjektet, noe som gjør at også risikofordeling og kontraktsbrudd må reguleres. Innenfor kontraktens rammer skal leverandøren oppnå fortjeneste, mens kunden skal oppnå å få levert riktig ting til riktig tid.[247] Reguleringen av kontraktsbrudd kan utelates i kontrakten og overlates til bakgrunnsretten. Likevel kan det være hensiktsmessig å ta inn bestemmelsene i kontrakten hvis man forutsetter at partene ikke kjenner bakgrunnsretten på området.[248] Dette er nødvendig når kontrakten skal brukes aktivt i prosjektet som et styrende verktøy for alle

246 Jf. Krüger (1989) s. 737
247 Jf. Kaasen (1994) s. 33
248 Jf. Torvund (1997) s. 171

interessenter, slik at den skrevne kontrakten harmonerer med bakgrunnsretten.[249] Hvis bakgrunnsretten skal fravikes er det nødvendig å ta inn reguleringen i kontrakten.

En slik fravikelse kan bli nødvendig ved bruk av smidig kontrakt, både fordi det er lagt opp til et tett samarbeid mellom partene og fordi dette samarbeidet i seg selv er en forutsetning for å lykkes. I en slik kontrakt kan det stilles spørsmål ved om sanksjonene ved kontraktsbrudd skal være de som følger av bakgrunnsretten, eller om de må ha en annen form.[250] Videre kan det hevdes at formålet med slik samspillkontrakt må få større betydning ved tolkning og utfylling, enn den tradisjonelle bakgrunnsretten.[251] Derfor vil jeg i dette kapittelet se nærmere på kontraktsbrudd og forholdet til risikofordelingen i den smidige kontrakten.

3.7.1 Risikofordelingen ved bruk av smidig metode

I underkapittel 2.5.2 gikk jeg inn på hvordan fordelingen av *vederlagsrisikoen* i en kontrakt er et grunnleggende element for vurdering av om en oppfyllelsessvikt er et kontraktsbrudd eller ikke. Utgangspunktet i en tradisjonell kompleks tilvirkningskontrakt er at risikofordelingen følger *funksjonsdelingen* mellom partene. Det vil si at *årsaken* til oppfyllelsessvikten har betydning for eventuelle konsekvenser av svikten. Hvordan risikoen skal fordeles må alltid vurderes i det enkelte tilfelle, og det ligger i prosjektets natur at ingen prosjekter er like. Likevel er det hensiktsmessig å trekke frem ulike momenter og aspekter som må ligge til grunn for vurderingen i en konkret sak. Jeg har tidligere pekt på nødvendigheten av å kontraktsfeste utviklingsprosessen ved bruk av smidige metoder. Når for eksempel gjennomføringen ved bruk av Scrum er regulert, vil det være tydelig hvilke roller, og dermed hvilke parter, som har hvilke arbeidsoppgaver. I bestemmelsene brukes for eksempel begrepet «ansvar for», som ikke betyr noe mer enn at en oppgave eller aktivitet skal utføres av en rolle. Likevel vil en slik regulering være et tydelig utgangspunkt for funksjonsdelingen mellom partene. Dermed blir det lettere å finne *årsaken* til en oppfyllelsessvikt. Imidlertid er det flere forhold som kan påvirke dette utgangspunktet, og dermed forrykke risikofordelingen med utgangspunktet i funksjonsfordelingen. Disse forholdene kan være prinsippene bak samarbeidsmetoden, prismodellen, om det er en innsats- eller resultatforpliktelse og partenes subjektive forhold.

3.7.1.1 Samarbeidsmetoden

Jeg har tidligere vært inne på at et prosjekt som benytter smidige utviklingsmetoder kan ha en side til prosjekter som er basert på en såkalt *samspillmodell,* eller partneringmodell. Kontrakter basert på en samspillmodell er i økende grad

249 Se Haapio (2013) s. 62-63
250 Jf. Tvarnø (2002b) s. 80
251 ibid. s. 81

tatt i bruk i blant annet entrepriseprosjekter.[252] Det finnes ingen klar definisjon av en samspillkontrakt, men prosjektet skal gjennomføres basert på tillit, åpenhet, felles målsettinger og deling av risiko. Denne tankegangen ligger også til grunn for den proaktive jussens tilnærming til kontrakter. Utgangspunktet er også at det som skal tilvirkes ikke er klart definert ved kontraktsinngåelsen, men at et tett samarbeid øker kvaliteten og produktiviteten, reduserer kostnadene og gir muligheter for økt inntjening.[253] Etter min mening bør en smidig kontrakt ansees som en samspillkontrakt, selv om ikke det er tatt inn en egen bestemmelse om at prosjektet skal gjennomføres med samspillmodell. Realiteten i et prosjekt som benytter smidige utviklingsmetoder, ligger tett opp til en samspillmodell. I EBAs veileder for samspillentreprise er det tatt inn et forslag til utforming av en samspillkontrakt. Pkt 1.1 har følgende tekst:[254]

> «Prosjektet skal gjennomføres som samspillentreprise. Byggherre og Entreprenør forplikter seg med dette til åpent, ærlig og tillitsfullt samarbeid.»

Denne bestemmelsen gir partene neppe noen andre plikter enn det som jeg tidligere har lagt frem som bestemmelser i en smidig kontrakt. Men spesielt lojalitetsplikten vil nok få større betydning i en slik samspillkontrakt. Det er noe jeg også har påpekt tidligere, ved å legge inn i en eksplisitt bestemmelse om lojalitetsplikt i den smidige kontrakten. I en slik kontrakt vil kravet til lojalitetsplikten skjerpes.[255] Illojalitet når det gjelder samarbeid og medvirkning kan i seg selv bli sanksjonert som kontraktsbrudd.[256] Lojalitetsplikten kan også få betydning når sanksjoner for andre kontraktsbrudd skal avgjøres. I Rt. 2006 s. 999 hadde selger av fast eiendom brukt for lang tid på å vurdere om han ville benytte seg av retting etter kjøpers reklamasjon. Høyesterett påpeker at: «Selgerens rett etter § 4-10 til å møte et krav om prisavslag eller heving med retting må utøves under hensyntagen til den gjensidige lojalitetsplikten i kontraktsforhold.»[257] På denne bakgrunnen kan lojalitetsplikten også få større betydning i samspillkontrakter som ikke benytter strenge preklusjonsregler.

I tillegg til et skjerpet lojalitetskrav i samspillkontrakter, er det også grunn til å se generelt på risikofordelingen. Programvareutvikling er av natur en tilvirkning med stor usikkerhet, og dette er en av de viktigste årsakene til at smidige utviklingsmetoder benyttes. Usikkerheten håndteres til dels på bakgrunn av utviklingsprosessen, men også fordi kontraktsgjenstanden ikke er uttømmende spesifisert ved kontraktsinngåelsen. Dette betyr at kunden har mulighet

252 Jf. Bergsaker (2010) s. 170
253 Jf. Tvarnø (2002b) s. 75
254 Se Entreprenørforeningen – Bygg og Anlegg (2013) s. 25
255 Jf. Nazarian (2007) s.309
256 Jf. Haaskjold (2013) s. 83
257 Se Rt. 2006 s. 999 (avsnitt 46)

til å påvirke resultatet underveis i tilvirkningen, og det i seg selv gjør at kunden må bære en større del av risikoen. Leverandøren er likevel den parten som sitter med fagkunnskapen og har muligheten til å realisere effektmålene i utviklingsarbeidet. Denne muligheten har leverandøren fordi han har valgfrihet med tanke på hvilke funksjoner som skal utvikles og hvordan disse best mulig kan bidra til å nå målene. Denne gjensidige avhengigheten mellom partene forsterkes, som tidligere nevnt, ved at de fleste beslutninger underveis i prosjektgjennomføringen gjøres enten av scrumteamet (som består av både kunden og leverandøren), eller i gruppen for felles prosjektstyring. Disse forholdene gjør at utgangspunktet med fordeling av risiko basert på funksjonsdelingen kan være misvisende som grunnlag når årsaken til problemene skal avklares. Utfordringene med å utlede risikoforholdene som grunnlag ved kontraktsbrudd kan oppsummeres ved Tvarnø: [258]

> «Ved at opstille en kontrakt, som ikke eksplicit definerer produktet, men i stedet lægger fokus på processen og samarbejdet, bliver det vanskeligere at udlede retsfaktum og retsfølge.»

3.7.1.2 Prismodellen

En måte å regulere risikofordelingen på, er valget av prismodell. Sandvik påpeker at det i kontrakter hvor omfanget ikke er endelig definert ved inngåelsen «[…] naturligvis ligger nærmest for hånden å la arbeidet utføres som 'regningsarbeid', slik at en på denne måten direkte influerer på risikofordelingen».[259] Føyen mfl. sier at valget mellom fast pris og regningsarbeid «kan sies å avgjøre hvem av partene som skal bære vederlagsrisikoen».[260] I underkapittel 3.6.2 konkluderte jeg med at en prismodell basert på regningsarbeid er egnet for en smidig kontrakt. Denne prismodellen medfører at kunden bærer risikoen for at usikkerheten i prosjektet gjør at timeforbruket overstiger det partene hadde regnet med ved kontraktsinngåelsen. Hvis leverandørens evne til å levere avtalt innhold i et inkrement svikter, går tiden. Kunden må likevel betale. Jeg trakk frem at timeforbruket har en yttergrense med tanke på uforsvarlig forhold hos leverandøren.

Et annet spørsmål er om en angitt kostnadsramme eller kostnadsoverslag i kontrakten setter noen grenser for leverandørens krav om betaling for utviklingen. I de fleste tilfeller har kunden et budsjett som setter rammer for prosjektgjennomføringen. Derfor vil ofte kostnadsrammer eller kostnadsoverslag være tatt inn i kontrakten. For å ta stilling til om slike rammer får betydning for hva leverandøren kan kreve betaling for, må det gjøres en konkret tolkning av bestemmelsene. Utgangspunktet for tolkningen vil være kontraktens ordlyd, men

258 Se Tvarnø (2002a) s. 150
259 Se Sandvik (1966) s. 123
260 Se Føyen (2006) s. 446

er denne uklar må andre momenter trekkes inn. Et moment er hvor detaljert spesifikasjonen av programvaren er. Hvis spesifikasjonen er så detaljert at det kan danne grunnlag for en fast pris, kan det tale for at kostnadsoverslaget i utgangspunktet er bindende.[261] Motsatt vil uklare spesifikasjoner basert på usikre estimater tale for at overslaget ikke blir ansett for være bindende. Et eksempel som illustrerer dette, er en lagmannsrettsdom hvor leverandøren ikke ble holdt ansvarlig for kostnadsoverskridelser, fordi overslaget var basert på «[…] et altfor spinkelt grunnlag.»[262] På en annen side kan leverandørens rett til vederlag bli avgrenset hvis kostnadsoverslaget er basert på en uaktsom beregning.[263] I standardkontrakten PS2000 SOL er det tatt inn bestemmelser om at hvis faktisk kostnad overstiger det dobbelte av godkjent estimat, skal det ansees som kontraktsbrudd, jf. pkt. 10.1.3. Sanksjoner for dette er heving og erstatning.

Som jeg tidligere har vært inne på er estimatene basert på epos og brukerhistorier som regel usikre. Derfor kan det bli vanskelig å anse et kostnadsoverslag i en smidig kontrakt for bindende. I tillegg vil kundens aktive medvirkning og styring når det gjelder prioriteringer, avklaringer og endringer i produktkøen og sprintbackloggen får betydning for de løpende kostnadene. Tilleggsarbeid og endringsarbeid vil i en fastpriskontrakt medføre rett til tilleggsvederlag for leverandøren. Når det er regningsarbeid som utføres, blir det vanskeligere å avgjøre hvilke deler av betalingen som kan tilskrives tillegg eller endringer som får betydning når det gjelder kostnadsoverslaget.

Siden regningsarbeid endrer risikofordelingen mellom partene, oppstår det et spørsmål om dette får betydning for vurdering av kontraktsbrudd knyttet til forsinkelse og mangler. Sandvik mener at det ikke får betydning for kontraktsbrudd med tanke på forsinkelse av avtalte tidsfrister.[264] Grunnen til dette er at leverandøren er sikret vederlag for den ekstra innsatsen han må gjøre for å nå fastsatte frister. Når det gjelder mangelsspørsmålene er det mer tvilsomt. Når eventuelle sanksjoner for mangler skal vurderes, må det tas i betraktning at leverandøren kunne oppnådd betaling for det manglende arbeidet som førte til oppfyllelsessvikten.[265] Jeg kommer nærmere tilbake til disse forholdene under drøftelsen av sanksjoner i underkapittel 3.7.5.

Ved bruk av målpris blir situasjonen noe annerledes. Målprisen tar utgangspunkt i et samlet estimat og forutsetter en uttømmende spesifikasjon ved oppstarten av prosjektet.[266] Endringer som påvirker målprisen må håndteres med endringsprosedyrer slik at målprisen kan justeres. Et påslag for uforutsette kostnader kan også legges inn i målprisen, og ved bruk av en samspillmodell vil

261 Jf. Sandvik (1966) s. 174
262 Se RG 1986 s. 89 (s. 95)
263 Jf. Krüger (1989) s. 162
264 Jf. Sandvik (1966) s. 186
265 Jf. Sandvik (1966) s. 186
266 Se nærmere om målpris i underkapittel 3.6.2

eventuelle overskridelser deles mellom partene.[267] En slik løsning gjør at risikoen er mer tydelig fordelt på hver av partene.

3.7.1.3 Innsats- eller resultatforpliktelse

For å kunne ta stilling til om forsinkelse eller mangel er kontraktbrudd, er det et grunnleggende utgangspunkt å se på hva slags type ytelse leverandøren har påtatt seg å levere. Hvis leverandøren har påtatt seg å levere et bestemt resultat i henhold til spesifikasjoner og krav fra kunden, en resultatforpliktelse, kan avvik mellom levert resultat og kundens berettiget forventning til resultatet utgjøre et kontraktsbrudd. Motsatt vil en innsatsforpliktelse for leverandøren bestå i å levere tilstrekkelig faglig innsats for å nå kundens ønskede resultat. En mangel i disse tilfellene kan for eksempel være hvis leverandørens ressurser ikke yter forsvarlig faglig innsats. Skillet mellom innsats- og resultatforpliktelse kan også avgjøre om det er et ansvarsgrunnlag som kan gi rett til erstatning.[268] For å kunne avgjøre hva slags type ytelse en kontrakt omhandler, må det gjøres en konkret tolkning av kontrakten. Et prosjekt som gjennomføres med en smidig metode, og spesifikasjonen av leveransen kun inneholder krav om antall konsulenter, timeantall og kompetanse, må ansees for å være en innsatsforpliktelse. Dette vil typisk være en form for bistandsavtale eller rammeavtale, og kunden har risikoen for at et bestemt resultat nås.[269]

Hvis kontrakten inneholder en spesifikasjon av hva som skal utvikles, enten en uttømmende spesifikasjon eller en grov spesifikasjon, vil leverandøren kunne ansees for å ha påtatt seg en resultatforpliktelse.[270] Bestemmelser i kontrakten kan si noe om hva slags type forpliktelse leverandøren har påtatt seg. Den danske standardkontrakten K03 er et eksempel på det. Ifølge kontraktens pkt. 3.2.2 har leverandøren en resultatforpliktelse for det kontrakten omtaler som «Absolutte Krav». For andre skal leverandøren «bestræbe sig på at levere så mange af de Øvrige Krav som muligt». Bestemmelsen må tolkes som at leverandøren har en innsatsforpliktelse for disse øvrige kravene. Smidigavtalen SSA-S har, som tidligere drøftet, et skille mellom funksjonelle og ikke-funksjonelle krav. De ikke-funksjonelle kravene skal være uttømmende spesifisert ved kontraktsinngåelsen. Likevel skiller ikke avtalen mellom innsatsforpliktelse for noen krav, og resultatforpliktelse for andre krav. Avtalen har blant annet følgende bestemmelser om leverandørens leveranseplikt i pkt. 5.1, første avsnitt:

267 Se Bergsaker (2010) s. 175
268 Jf. Haaskjold (2013) s. 460
269 Se Torvund (1997) s. 70
270 Jf. Føyen (2006) s. 234

> «Leverandøren har ansvar for at leveransen dekker behovs – og løsningsbeskrivelsen i bilag 1 og 2 med eventuelle endringer i bilag 9, likevel slik at detaljspesifikasjonen og godkjenningskriteriene skal utgjøre den endelige spesifisering av hva som skal leveres for de områdene de dekker.»

Bestemmelsen er ikke like klar som tilsvarende i den danske K03, men den kan tolkes slik at leverandøren har et resultatansvar for alt som utvikles.

Når en bestemmelse om leveranseplikten er uklar, må det gjøres en tolkning av kontrakten for å avgjøre hva slags type forpliktelse leverandøren har påtatt seg. Noen retningslinjer for vurderingene gir Unidroit Principles of International Commercial Contracts 2010 (Unidroit) art. 5.1.5. Artikkelens pkt. d)[271] omhandler i hvilken grad kunden kan påvirke ønsket resultat, og kan være relevant ved bruk av en smidig metode. Likevel har leverandøren ansvar for å bistå på kundens ansvarsområder, fordi leverandøren er den som sitter på fagkunnskapen. Etter mitt syn må artikkelen pkt. d) tolkes slik at den gjelder der hvor kunden også sitter på fagkunnskapen og skal bidra i utviklingsarbeidet. Artikkelens pkt. c)[272] kan også være relevant – spesielt ved programvareutvikling som i seg selv ofte er knyttet til stor usikkerhet og risiko. Smidige metoder tas nettopp i bruk fordi usikkerheten er stor. Jeg mener likevel punktet må tolkes slik at dette gjelder prosjekter som har spesiell stor usikkerhet knyttet til muligheten for å oppnå ønsket resultat. Eksemplet Unidroit bruker er risikoen i et prosjekt hvor målet er å få en satellitt i bane, hvor feilraten erfaringsmessig er på 22 prosent. Slike prosjekter vil komme i den kategorien som Føyen mfl. omtaler som «de mer forskningspregede avtaler», og som må betegnes som innsatsforpliktelser.[273] I slike prosjekter vil det trolig komme frem av kontrakten at det stor usikkerhet knyttet til om ønskede resultater kan nås. Noe som i seg selv er et tolkningsmoment som Unidroit pkt. 5.1.5, a)[274] også beskriver.

Et eksempel er en tingrettsdom hvor det skulle utvikles et nettbasert regnskapsprogram der liknende løsninger ikke tidligere hadde vært laget.[275] Leverandøren hadde gitt uttrykk for at det var stor usikkerhet knyttet til resultatet, og ville derfor ikke påta seg oppdraget til fast pris eller definert tidsplan. Det var heller ikke utarbeidet en definert kravspesifikasjon. Retten konkluderte med at dette var en innsatsforpliktelse. Konklusjonen etter dette er at en smidig kontrakt med en viss spesifikasjon av kontraktsgjenstanden, regulert utviklingsprosess og et normalt usikkerhetsnivå, i utgangspunktet må ansees som en resultatforpliktelse for leverandøren.

---

271 «(d) the ability of the other party to influence the performance of the obligation»

272 «(c) the degree of risk normally involved in achieving the expected result»

273 Jf. Føyen (2006) s. 44-45

274 «(a) the way in which the obligation is expressed in the contract»

275 Se TOSLO-2001-11218

3.7.1.4 Partenes subjektive forhold

Partenes subjektive forhold dreier seg om at en parts handlinger eller unnlatelser er uforsvarlige. I så fall er det et grunnleggende utgangspunkt at parten selv må bære skadefølgene, og at dette vil kunne påvirke risikofordelingen.[276] Hvis det for eksempel oppstår en oppfyllelsessvikt hvor kunden bærer risikoen fordi prismodellen er basert på regningsarbeid, kan subjektive forhold hos leverandøren føre til at det er leverandøren selv som likevel må bære risikoen. Tidligere har jeg for eksempel trukket frem betydningen av påløpte kostnader som skyldes uforsvarlige forhold hos leverandøren, jf. underkapittel 3.6.2.

Det stilles strenge krav til aktsom opptreden i kontraktsforhold som gjelder næringsforhold.[277] De strenge kravene er basert på at partene frivillig har påtatt seg pliktene sine. I tillegg har leverandøren påtatt seg oppdraget som en profesjonell part mot betaling. Dette profesjonsansvaret strekker seg langt, men kunden kan ikke kreve at det alltid er den optimale løsningen som velges. Leverandøren har et visst spillerom ved valg av løsninger. I Rt. 1995 s. 1350 ble det slått fast at leverandøren må ha «[…] et visst spillerom før atferd som kan kritiseres, må anses som erstatningsbetingende uaktsomhet.»[278] Det er kunnskapen om valgte løsning på utførelsestidspunktet som er avgjørende i vurderingen, jf. Rt. 1968 s. 783.[279] Den raske teknologiutviklingen gjør at det på noen områder ikke etablerer seg normer som holder seg over tid. Derfor kan det være en utfordring å avgjøre om leverandørens valgte løsning er i tråd med en bransjenorm. Likevel må kunden kunne forutsette at leverandøren holder seg oppdatert på sitt eget fagfelt, og derfor velger løsninger som er i takt med den teknologiske utviklingen.[280]

Profesjonsansvaret er også knyttet til en abstrakt mangelsvurdering med tanke på hvilke krav som kan stilles til kvaliteten på leveransen. Når det gjelder krav til leveransens kvalitet, se avsnitt 3.7.4.4, vil det i utgangspunktet ikke være nødvendig også å ta stilling til om leverandøren har utført arbeidet med tilstrekkelig aktsomhet. På grunn av funksjonsdelingen bærer leverandøren likevel risikoen for kvaliteten, men det kan styrke kundens stilling når mangelsvurderingen skal gjøres.[281] I tillegg vil uaktsomhet kunne utløse krav om erstatning som en supplerende sanksjon. For eksempel i en lagmannsrettsdom, hvor entreprisen var basert på regningsarbeid, fikk byggherren dekket merkostnadene for mangler ved arbeidet.[282] Lagmannsretten konstaterte at «[…] feil som følge av

276 Jf. Sandvik (1966) s. 296
277 Jf. Hagstrøm (2011) s. 468
278 Se Rt. 1995 s. 1350 (s. 1356)
279 Se Rt. 1968 s. 783 (s. 788)
280 Se Torvund (1997) s. 205
281 Jf. Sandvik (1966) s. 296
282 LB-2004-11707

manglende aktsomhet, f.eks. slurv ved entreprenørens utførelse, […] har […] byggherren rett til å gjøre mangelsbeføyelser gjeldende».

Den skjerpede lojalitetsplikten i en smidig kontrakt, jf. avsnitt 3.7.1.1 og varslingsplikten, se avsnitt 3.7.2.1, er tett knyttet til en aktsom opptreden. Normalt må partenes opptreden være klanderverdig for at den skal regnes som illojal.[283] Illojalitet kan sanksjoneres som kontraktsbrudd, og det legger et press på partene om å oppfylle pliktene i kontrakten.

3.7.2 Leveranser, forsinkelse og mangler

Den parten som har risikoen for årsaken til en oppfyllelsessvikt kan bli utsatt for sanksjoner. I forrige underkapittel drøftet jeg forhold knyttet til risikofordelingen. I dette og de neste underkapitlene skal jeg gå nærmere inn på vurderingen av oppfyllelsessvikt, nærmere bestemt forsinkelser og faktiske mangler. Rettsmangler vil ikke bli behandlet.

Et grunnleggende aspekt ved bruk av smidige utviklingsmetoder, er at denne måten å gjennomføre et prosjekt på, er risikoreduserende med tanke på usikkerheten ved programvareutvikling. Uforutsette problemer blir raskt avdekket ved bruk av sprinter, og nødvendige tiltak kan iverksettes av scrumteamet og gruppen for felles prosjektstyring. Dette gjør at indikasjoner på at leveransene ikke svarer til kundens berettigede forventninger avdekkes før det kan konstateres forsinkelser eller mangler. Problemer som ikke like lett lar seg korrigere kan håndteres med prosjektintegrert mekling. I tillegg vil partenes løpende adgang til oppsigelse av kontrakten være en mulig for å komme seg ut av problemer som ikke lar seg løse ved hjelp av meklingen. Til tross for disse utgangspunktene kan det være behov for å ha et avklart forhold til kontraktsbrudd og eventuelle sanksjoner. En oppsigelse fra en av partene kan medføre både økonomiske og praktiske konsekvenser for både kunden og leverandøren. Da kan bestemmelser om kontraktsbrudd være alternative mekanismer som tar sikte på å opprettholde balansen i kontrakten, samt gi insentiver til fortsettelse av samarbeidet.

Utgangspunktet er at forsinkelse og mangler er knyttet til avtalte leveranser. I en smidig kontrakt skjer det forløpende leveranser ved hver sprint. Det vil neppe være hensiktsmessig å gjøre en juridisk vurdering av eventuell oppfyllelsessvikt av hvert inkrement. For det første vil den vurderingen og testingen som gjøres ved en sprintreview ikke være så omfattende som den bør være for programvare som skal settes i produksjon. For det andre vil som regel inkrementet være avhengig av funksjonalitet som utvikles i andre inkrementer, og derfor vil ikke inkrementet egne seg for en vurdering med tanke på kontraktsbrudd. Et viktig prinsipp ved bruk av smidige metoder er at det skal utvikles programvare som kan settes i produksjon underveis i prosjektgjennomføringen. Derfor skal det utarbeides en plan for delleveranser som er egnet til å produksjonsettes. Det

283 Jf. Nazarian (2007) s. 252

er disse delleveransene det er aktuelt å knytte til sanksjoner ved kontraktsbrudd. Også i tradisjonelle kontrakter for programvareutvikling er det vanlig å knytte sanksjoner til delleveranser.[284]

Når programvaren er leveringsklar fra leverandøren etter en sprint som er en delleveranse, betyr ikke det at programvaren er levert. Det er nødvendig å avsette tilstrekkelig tid slik at kunden kan gjennomføre *akseptansetesting* av programvaren med tanke på godkjennelse. Hvis kunden godkjenner delleveransen etter utløp av den avtalte testperioden, er programvaren å anse som levert. Akseptansetestingen innebærer at kunden må avdekke om den leveringsklare programvaren inneholder mangler, og hvis det er tilfelle foreligger det forsinkelse.[285] Dette betyr at kunden må gjøre en mangelsvurdering i testperioden på samme måte som det gjøres en mangelsvurdering etter levering. Manglene som avdekkes i testperioden «transformeres» til forsinkelse, og eventuelle sanksjoner er knyttet til forsinkelse som kontraktsbrudd.[286]

3.7.2.1 Varsling og reklamasjon

Med varsling mener jeg partenes opplysningsplikt overfor hverandre, altså plikten til å varsle om hindringer av egen oppfyllelse etter kontrakten. Denne plikten er en viktig praktisk regel underveis i prosjektgjennomføringen, og derfor ser jeg nærmere på denne. Reklamasjon er en annen form for varsling. Den brukes når en part vil iverksette sanksjoner overfor den andre parten på grunn av dennes kontraktsbrudd. Jeg vil også kort redegjøre for reklamasjonsreglenes utgangspunkt.

I kontraktsforhold som reguleres av kjøpsloven[287] fremkommer det en opplysningsplikt for leverandøren i §§ 28 og 40 (1), og for kunden i § 58. For kontraktsforhold om ikke reguleres av kjøpsloven kan en slik opplysningsplikt utledes av partenes lojalitetsplikt,[288] og med bakgrunn i Rt. 1970 s. 1059.[289] I denne dommen var ikke selger forpliktet til å levere på grunn av force majeure, men ble likevel erstatningsansvarlig fordi han «[…] på et tidligere tidspunkt burde ha underrettet […] [kjøper] om faren for forsinkelse».[290] Det er viktig at partene varsler hverandre hvis en part ikke klarer å oppfylle sine plikter etter kontrakten. Begge parter er interessert i at slike forhold blir avdekket så raskt som mulig, slik at nødvendige tiltak kan settes i gang for å begrense virkningene.[291] SSA-S har regler for leverandørens og kundens varslingsplikt i pkt. 11.2 og pkt.

284 Jf. Føyen (2006) s. 459
285 Jf. Føyen (2006) s. 469
286 Se Torvund (1997) s. 191
287 Kjøpsloven (1988)
288 Jf. Nazarian (2007) s. 210
289 ibid. s. 531
290 Se Rt. 1970 s. 1059 (s. 1065)
291 Jf. Haaskjold (2013) s. 653

12.2. Varslingene gjelder alle forhold som gjør at partene ikke kan oppfylle sine plikter etter avtalen. PS2000-kontraktene har ingen bestemmelser om varslingsplikt for partene. Som nevnt vil uansett en slik varslingsplikt gjelde på grunn av den gjensidige lojalitetsplikten. Ved for eksempel bruk av Scrum, vil det tette samarbeidet gi partene rikelige muligheter for gjensidig varsling.

Et spørsmål er om det har betydning om varslingen skjer skriftlig eller ikke. Spesielt gjelder dette ved bruk av smidige utviklingsmetoder hvor mye av kommunikasjonen mellom partene er uformell. Av notoritetshensyn bør naturligvis varslingen skje skriftlig, men prinsippet om formfrihet må gjelde også slike varsler. SSA-S' bestemmelser om varsling stiller krav til skriftlighet, og hva varselet skal inneholde. Tilsvarende skriftlighet har bestemmelsen om leverandørens varsling om forsinkelse i IKT 2010, jf. pkt. 6.8.1. Ingen av bestemmelsene har regler om konsekvensene av at varselet ikke gis skriftlig. Etter mitt syn bør kravet til skriftlig varsel sees på som en bevisbyrderegel, på samme måte som ved reklamasjon.[292]

Reklamasjon er for eksempel et varsel fra kunde til leverandør om at det er en mangel i levert programvare, som av kunden ansees som et kontraktsbrudd. Varselet forteller også at kontraktsbruddet vil bli sanksjonert. I forbindelse med reklamasjon er det også en reklamasjonsplikt. Denne plikten handler om at den parten som reklamerer må gjøre dette innen visse frister, hvis ikke vil han tape retten til sanksjoner.[293] Reklamasjon skjer ved levering eller etter levering. Før levering er det varslingsplikten som skal ivareta partenes muligheter for tiltak mot eventuell oppfyllelsessvikt. Unntaket fra dette er ved reklamasjon for antesipert kontraktsbrudd, som jeg for øvrig ikke skal behandle.

Utgangspunktet for reklamasjonsplikten er en absolutt og en relativ reklamasjonsfrist. For kontrakter som kjøpsloven[294] regulerer, er den absolutte fristen to år, jf. § 32 (2), mens den relative fristen er at kjøperen må gjøre gjeldende en mangel «[…] innen rimelig tid etter at han oppdaget eller burde ha oppdaget den», jf. § 32 (1). For eksempel for en smidig kontrakt, som ikke reguleres av kjøpsloven, finnes det ingen ulovfestet absolutt reklamasjonsfrist i bakgrunnsretten.[295] Når det gjelder relative reklamasjonsfrister er det mer uklart, men prinsippene bak relative reklamasjonsfrister i kontraktslovene, kan brukes i kontrakter som ikke er lovregulerte.[296] I de fleste kontrakter for programvareutvikling vil ikke disse problemstillingene være aktuelle, siden reklamasjonsfrister og garantiperioder er uttømmende regulert. Det er som regel ønskelig for leverandøren å avklare når ansvaret for leveransen er avsluttet.[297]

292 Se Simonsen (1999) s. 323-324
293 Jf. Monsen (2010) s. 148
294 Kjøpsloven (1988)
295 Se Monsen (2010) s. 184
296 Jf. Hagstrøm (2011) s. 353
297 Jf. Føyen (2006) s. 462

I en smidig kontrakt er det reklamasjon knyttet til delleveransene som er aktuelt. I perioden med akseptansetesting skal kunden undersøke og teste delleveransen. Det er ved utgangen av denne perioden det kan være aktuelt med reklamasjon. Etter testing og godkjenning er programvaren levert. Det kan imidlertid være vanskelig å oppdage alle mangler i testperioden, og ofte kan problemer vise seg først når systemet er tatt i bruk. Derfor har standardkontraktene en feilrettingsperiode eller garantiperiode som har absolutte frister, men før utløpet av perioden er det en relativ reklamasjonsfrist.

IKT 2010 har en feilrettingsperiode som skal avtales i bilag, jf. pkt. 10, og tilsvarende skal det avtales en garantiperiode i PS2000-kontraktene, jf. pkt. 3.5.4. SSA-S har både en godkjenningsperiode på tre måneder, jf. pkt. 2.5.2, og en garantiperiode som i utgangspunktet er på tre måneder, jf. pkt. 4.1. Forskjellen mellom godkjenningsperioden og garantiperioden i SSA-S er hovedsakelig to forhold. Det ene er at kunden kan stoppe den løpende godkjenningsperioden underveis, hvis man finner feil som gjør det umulig å bruke systemet (eller deler av systemet). Perioden starter ikke igjen før leverandøren har rettet feilene, jf. pkt. 2.5.4. Garantiperioden har ikke en slik regel, og grunnlaget for reklamasjonsretten er strengere. Det er bare mangler som ikke burde vært oppdaget under akseptansetesting eller i godkjenningsperioden, som leverandøren skal rette uten ekstra kostnader, jf. pkt. 4.4. Etter utløpet av de absolutte fristene i kontraktene kan ikke kunden reklamere. I underkapittel 3.7.6 ser jeg nærmere på forholdet til garanti og etterfølgende vedlikehold i smidige kontrakter.

3.7.3 Forsinkelser

Det er leverandørens forsinkelser med tanke på avtalt tid for delleveranser, som er aktuelt i denne sammenheng. Partene skal utarbeide en fremdriftsplan, der det fremgår når de ulike delleveransene skal leveres. I en tradisjonell kontrakt, hvor pris, tid og omfang er faste størrelser ved kontraktsinngåelsen, vil milepælsplanen være et vanlig utgangspunkt for å vurdere forsinkelser. Som jeg tidligere har trukket frem, er det en stor utfordring å estimere tid og omfang. Likevel vil det i noen typer prosjekter være nødvendig å kunne levere programvare med en viss funksjonalitet til en fastsatt dato. Denne datoen vil dermed være styrende for planleggingen og prioriteringen underveis i prosjektgjennomføringen. I slike tilfeller vil kunden være avhengig av at det avtales tydelige milepæler for flere delleveranser, som kan bli sanksjonert ved forsinkelser. I andre typer prosjekter er funksjoner og egenskaper ved programvaren viktigere enn levering til en fast dato.

I tillegg til at avtalte milepæler for delleveransene er en forutsetning for at en forsinkelse kan vurderes som kontraktsbrudd, må også årsaken til forsinkelsen ligge hos leverandøren. Med tanke på funksjonsdelingen i kontrakten kan det være at forsinkelsen skyldes forhold kunden har ansvaret for. Dermed er det

ikke aktuelt å anse forsinkelsen som kontraktsbrudd fra leverandørens side. Dette viser hvor viktig det er at prosjektgjennomføringen og utviklingsprosessen er regulert i kontrakten, som beskrevet i underkapittel 3.6.1. En slik kontraktsregulering klargjør hvilke plikter og ansvarsområder partene har, og årsakssammenhengen mellom forsinkelse og ansvarsforhold kommer tydelig frem.

Det er vanlig å ha bestemmelser om fristforlengelse for partene hvis årsaken til forsinkelsen ligger hos den andre parten, eller som følge av endringer, se for eksempel NS 8407, pkt. 33.1 og pkt. 33.2. Uten en slik bestemmelse, vil et ønske om fristforlengelse kunne behandles med avtalte endringsprosedyrer. Uansett vil ikke en forsinkelse kunne sanksjoneres av den parten som er årsaken til forsinkelsen. Likevel vil en bestemmelse om fristforlengelse kunne fungere som et verktøy for partene til å finne felles optimal løsning, før man konstaterer et kontraktsbrudd. SSA-S har en slik generell bestemmelse for leverandøren i pkt. 11.3. I en smidig kontrakt bør det legges til rette for fristforlengelse for begge parter. Hvor praktisk en slik bestemmelse vil være er usikkert, da varslingsplikten om forsinkelse vil utløse tiltak tidligere i prosessen. I en smidig kontrakt bør det ikke legges opp til å endre på sprintenes lengde, fordi det vil forstyrre utviklingsmetoden. Konsekvensene av en tilleggsfrist i en smidig kontrakt vil være å flytte en delleveranse slik at den inneholder ekstra sprint(er). Alternativt kan datoen for delleveranse stå fast, men scrumteamet fjerner elementer fra sprintbackloggen. Det gjør at delleveransen kan leveres som planlagt, men med mindre funksjonalitet. Uansett må partene gjøre en konkret vurdering for å finne best mulig løsning.

Prismodellen med regningsarbeid gjør at det er kunden som i utgangspunktet har risikoen for merkostnader for uforutsett arbeid. Derfor kan leverandøren forsere forsinkelsen uten å måtte betale for merkostnadene. Imidlertid er det sjelden praktisk, på kort sikt, å trekke inn flere ressurser for å rekke en tidsfrist fordi nytt personell trenger tid på å sette seg inn i prosjektet. Ellers kan leverandøren pålegge overtidsarbeid for eksisterende ressurser, slik at en mindre forsinkelse kan unngås. Over lenger tid er bruk av overtidsarbeid lite heldig, og forskning viser at produktiviteten i utviklingsteamet går ned.[298] Hvis det er tendenser til stadige forsinkelser, må partene endre arbeidsmengden som legges inn i sprintbackloggen, og justere fremdriftsplanen.

3.7.4 Mangelsvurderingen

I henhold til kjøpsloven[299] skal ytelsen «være i samsvar med de krav til art, mengde, kvalitet, andre egenskaper og innpakning som følger av avtalen», jf. § 17 (1). SSA-S pkt. 11.1 slår blant annet fast at det foreligger mislighold fra leve-

298 Jf. Cohn (2010) s. 290-291

299 Kjøpsloven (1988)

randøren hvis «[…] leveransen ikke er i samsvar med de funksjoner og krav og frister som er avtalt». Liknende bestemmelse finnes i PS2000-kontraktene i pkt. 6.1.2: «Leveransen har Feil eller mangler dersom det foreligger avvik mellom avtalt og levert Leveranse […]». Det er altså de avtalte egenskapene til ytelsen som er utgangspunktet for mangelsvurderingen. Avtalte egenskaper er hva som er spesifisert ved kontraktsinngåelsen, samt partenes avtalte etterfølgende endringer og detaljspesifiseringer. Ytelsens faktiske egenskaper subsumeres under kontraktens bestemmelser. Hvis de avtalte egenskapene er uklare og lite konkrete, må kravene til ytelsen avgjøres ved tolkning og utfylling av kontrakten.[300] Det er det konkrete mangelsbegrepet som kan utledes av ytelsens avtalte egenskaper. Likevel kan det være nødvendig å supplere mangelsvurderingen med et abstrakt mangelsbegrep. Spesielt hvis kontrakten ikke har tilstrekkelige holdepunkter for krav til ytelsen.[301] Kjernen i mangelsvurderingen er å komme frem til hvilke berettigede forventninger kunden kan ha i henhold til kontrakten.[302]

3.7.4.1 Forskjellen mellom feil eller mangler

Ved noen tilfeller blir begrepet «feil og mangler» brukt, mens andre ganger kun «mangler». Føyen mfl. trekker frem at det er en grense mellom «feil» og «mangler».[303] Grunnen til å trekke frem en slik grense, er at all programvare, i større eller mindre grad, vil inneholde feil. Noen feil kan være av ubetydelig karakter, mens andre feil er så alvorlige at kunden neppe kan leve med dem. På en annen side, kan flere mindre feil til sammen forringe leveransens kvalitet. Etter mitt syn vil det ikke være nødvendig å trekke en slik grense. I den konkrete vurderingen handler det i praksis ikke om et forhold er en feil eller en mangel, men om omfanget og karakteren til feilene og manglene.[304]

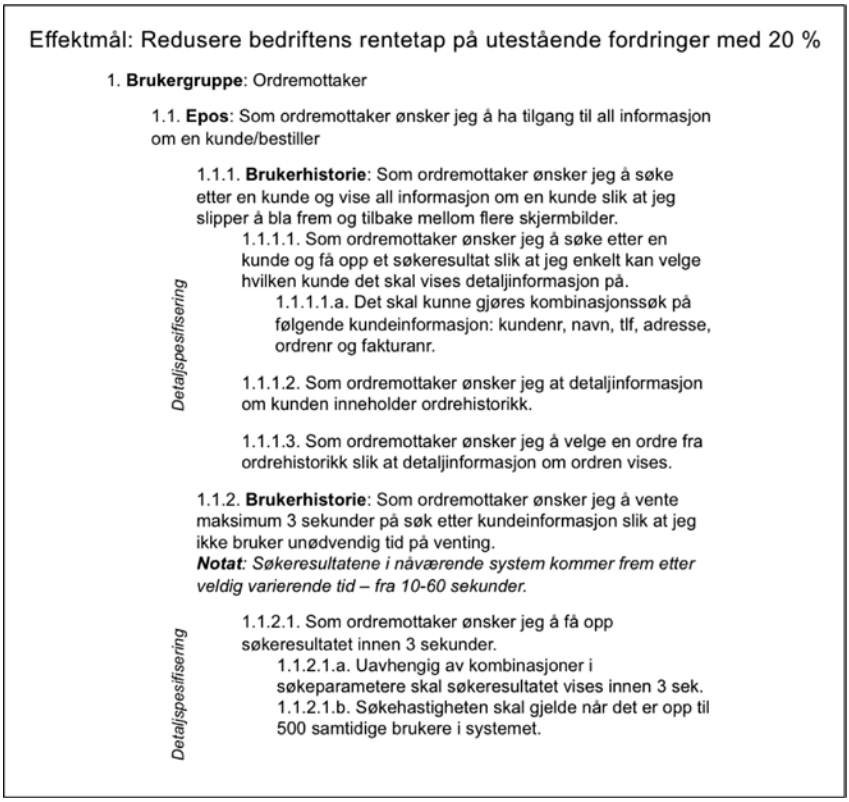
3.7.4.2 Inkrementer og delleveranser

Resultatet etter hver sprint er altså et inkrement, som også kalles en delleveranse når milepæler for produksjonssetting er avtalt, jf. avsnitt 3.6.1.1. Det vil derfor være inkrementene som i praksis vil være gjenstand for en mangelsvurdering. Egenskaper for inkrementet blir avtalt mellom partene ved detaljspesifiseringen i sprintplanleggingen, og det som skal utvikles vil fremkomme av sprintbacklogen. I tillegg er det avtalt hvilke generelle krav som inkrementet skal oppfylle, se underkapittel 3.6.1. Noen av disse kravene er enkle å kontrollere, som for eksempel at dekningen på enhetstester er over en viss prosent, og at enhetstestene skal gjennomføres uten feil. Her kan imidlertid skillet mellom feil og mangler få en

300 Jf. Buch (2007) s. 23
301 Jf. Hagstrøm (2011) s. 166
302 Jf. Torvund (1997) s. 194
303 Se Føyen (2006) s. 478-480
304 Jf. Udsen (2013a) s. 570

viss betydning. Det kan tenkes at enkeltfeil som avdekkes av enhetstestene ikke har så stor betydning at inkrementet kan sies å ha en mangel. Dette må, som nevnt, vurderes konkret på bakgrunn av hvor mange slike feil det er, og betydningen av disse. Selv om feilene ikke er mangler, vil nok kunden likevel sørge for å rette feilene, ved at de blir lagt inn i sprintbackloggen for neste sprint.

Et annet vesentlig krav til et inkrement er det skal oppfylle alle akseptansekriteriene til brukerhistoriene som skal utvikles i sprinten. I figur 3.9 illustrerer jeg hvordan deler av et effektkart i praksis blir detaljspesifisert med akseptansekriterier. Figuren viser to brukerhistorier som er knyttet til et epos og en brukergruppe. På øverste nivå er effektmålet angitt. Produkteier vil bearbeide den initielle produktkøen, med for eksempel tilleggsinformasjon som merknader til en brukerhistorie. Videre vil partene i detaljspesifiseringen splitte opp en brukerhistorie i mindre brukerhistorier og tilhørende akseptansekriterier. For eksempel er brukerhistorien 1.1.1 splittet i tre nye brukerhistorier (1.1.1.1-1.1.1.3). Vi ser i figuren at også ikke-funksjonelle krav er detaljspesifisert under brukerhistorie 1.1.2. Hvis brukerhistorien 1.1.1.1 skal utvikles i en sprint, skal inkrementet etter sprinten oppfylle akseptansekriteriet i 1.1.1.1.a. Hvis kunden avdekker at det ikke går an å gjøre et kombinasjonssøk, men kun søk på et parameter av gangen, må det ansees som en mangel.



Figur 3.9: Eksempel på detaljspesifisering av to brukerhistorier

En mangel kan også være knyttet til avhengigheten mellom inkrementer. Det kan være avhengigheter mellom inkrementer som leveres fra flere team etter en sprint, og det kan være mellom inkrementer som til sammen utgjør en delleveranse. De største utfordringene er nok knyttet til delleveransene, som er ment for produksjonssetting. Da må alle inkrementer og komponenter som utgjør delleveransen fungere sammen. Etter siste sprint før en delleveranse, skal alle inkrementene spille sammen for første gang. Da er det ikke uvanlig at det oppstår problemer. Det at inkrementene ikke fungerer sammen må klart være en mangel.

Men, hvis de ulike inkrementene ikke fungerer sammen underveis, utenom delleveransene, er det mer tvilsomt om kan ansees som en mangel. Selv om det er et viktig prinsipp at det skal leveres programvare som virker ved hver leveranse, er ikke dette ment slik at alle komponenter skal kunne spille sammen etter

hver sprint.[305] Hvis det er definert akseptansekriterier forut for en sprint, som gjelder integrasjon med andre inkrementer, så må imidlertid inkrementet oppfylle integrasjonskriteriene. Det kan også oppstå avhengighet og integrasjonsproblemer mellom delleveranser, og slike feil må også ansees for å være mangler.

De avtalte kravene til inkrementet, sammen med innholdet i produktbackloggen, gir nokså tydelig utgangspunkt for mangelsvurderingen. Selv om inkrementet tilfredsstiller kravene og inneholder den funksjonaliteten som tilsynelatende er spesifisert som i figur 3.9, kan det godt være at kunden mener inkrementet ikke oppfyller de berettigede forventingene. Utfordringen er å avgjøre hvilke forventninger kunden kan ha når detaljspesifikasjonen ikke inneholder mer informasjon enn den gjør når utviklingen skjer ved smidige metoder. Når utgangspunktet for utviklingsarbeidet er en spesifikasjon som illustrert i figur 3.9, har leverandøren stor valgfrihet med tanke på hvordan systemet skal utvikles. Tilsvarende valgfrihet kan også totalentreprisekontrakter inneholde, slik det er forutsatt i NS 8407, pkt. 14.6. Den sier at entreprenøren innenfor kontraktens rammer har «rett til å velge hva slag materiale, utførelse og løsning han vil oppfylle kontrakten med».

Valgfriheten ved programvareutvikling er knyttet til blant annet hvordan den tekniske arkitekturen skal være, hvordan datamodeller og databasen skal struktureres og hvordan brukergrensesnittet skal utformes. Smidigtankegangen bygger nettopp på en slik valgfrihet, fordi leverandøren da har muligheten til å finne de mest optimale løsningene underveis i utviklingen. Foruten de avtalte elementene i sprintbackloggen og kravene til inkrementet, må andre holdepunkter trekkes inn i mangelsvurderingen. Når ikke spesifikasjonene er uttømmende, kan det være større grunn til å trekke inn en abstrakt mangelsvurdering.[306] Angitte formål med anskaffelsen er også av betydning for mangelsvurderingen. Både angitte formål og en abstrakt mangelsvurdering vil kunne begrense den valgfriheten leverandøren i utgangspunktet har.

3.7.4.3 Betydning av formålsangivelser

Angivelse av formålet har betydning for mangelsvurderingen.[307] I smidigavtalen SSA-S forutsettes det i pkt. 6.1 at kunden skal spesifiseres anskaffelsens formål i bilag 1. Tilsvarende bestemmelse har den danske K03 i pkt. 2.2. Det er en forutsetning at leverandøren kjenner til formålet hvis den skal få betydning for mangelsvurderingen. Ved bruk av effektkart i kontrakten, kommer formålene (effektmålene) tydelig frem. Disse effektmålene er overordnede, slik at det er anskaffelsen som helhet som skal være egnet til oppfylle effektmålet. I figur 3.9 er effektmålet at bedriftens rentetap på utestående fordringer skal reduseres

305 Se Cohn (2010) s. 262-263
306 Jf. Hagstrøm (2011) s. 172
307 Jf. Torvund (1997) s. 73

med 20 prosent. Det vil være vanskelig å vurdere om det enkelte inkrement er egnet som bidrag til å nå effektmålet. Mer aktuelt kan det derfor være å se på den spesifiserte hensikten med et epos eller en brukerhistorie. I nevnte figur trekker jeg frem følgende eksempel:

> Som ordremottaker ønsker jeg å søke etter en kunde og vise all informasjon om en kunde *slik at jeg slipper å bla frem og tilbake mellom flere skjermbilder.*

I denne brukerhistorien har jeg uthevet beskrivelsen av hensikten, eller formålet med brukerhistorien. Dette formålet, sammenstilt med målsettingen om redusert rentetap, tilsier at effektiviteten til ordremottakeren skal økes. Hvis leverandøren utvikler en funksjonalitet som ikke øker brukerens effektivitet, vil inkrementet inneholde en mangel. Et eksempel på bruk av effektmål er lagmannsrettsdommen om NSBs anskaffelse av et system for automatisk ruteopplysning.[308] Effektmålet i anskaffelsen var blant annet at: «Tjenesten skal erstatte nåværende manuelle og tastestyrte tjenester som gir opplysning om togtider og togforbindelser». Et av målepunktene for effektmålet var at opptil 90 prosent av samtalene skulle gjenkjenne oppgitt stasjon, dato og klokkeslett. Underveis i utviklingen viste det seg vanskelig for leverandøren å levere et system som gjenkjente mer enn 30-40 prosent av samtalene. Retten konkluderte med at ytelsen var mangelfull.

Barbo trekker frem et forhold er relatert til formålsangivelser i totalentreprise. Selv om formålet isolert sett er oppfylt, kan ytelsen likevel være mangelfull hvis leverandørens ytelse «[…] nedsetter byggets *anvendelighet* og *funksjon* i forhold til andre, nærliggende løsninger.» [309] Eksempelet han trekker frem fra en voldgiftsdom er at entreprenøren valgte en bygningskonstruksjon, som begrenset innredningsmuligheten for kunden. Slike forhold kan også overføres til programvareutvikling. Et praktisk eksempel kan være at leverandøren har utviklet en funksjon som oppfyller dens hensikt, ved at brukeren slipper å bla frem og tilbake mellom skjermbilder. Så viser det seg at funksjonen er så komplisert å bruke at ingen effektivitet er oppnådd, og programvaren er dermed mangelfull.

3.7.4.4 Abstrakt mangelsvurdering

Hvis ikke spesifikasjonene i kontrakten og tolkningen av disse gir tilstrekkelige holdepunkter for en konkret mangelsvurdering, må man supplere med en abstrakt mangelsvurdering. Utgangspunktet for en abstrakt mangelsvurdering i kjøpsforhold er at ytelsen skal være «vanlig god handelsvare».[310] I bustadoppføringsloven[311] § 7 er tilsvarende bestemmelse formulert som at entreprenøren

308 LB-2004-8893
309 Se Barbo (1990) s. 58
310 Jf. Rt. 1998 s. 774 (s. 781)
311 Bustadoppføringslova (1997)

skal «utføre arbeidet på fagleg godt vis». Loven gjelder forbrukerkjøp, men bestemmelsen er en kodifisering av bakgrunnsretten i entrepriserett. Generelt for tilvirkningskontrakter er at innholdet i et slikt begrep er knyttet til hva som «[…] er akseptert som god skikk og bruk i faget eller bransjen.»[312] I NS 8407 kommer dette frem i pkt. 14.5:

> «Er ikke annet avtalt, skal kontraktsgjenstanden være i overensstemmelse med Norsk Standard og for øvrig i samsvar med aksepterte normer på tilbudstidpunktet.»

Denne bestemmelsen sier både at arbeidet skal tilfredsstille krav i standarder utgitt av Norsk Standard, og at utførelsen skal være i overensstemmelse med begreper som «god byggeskikk» og «fagmessig».[313] Tilsvarende krav må også gjelde for programvareutvikling. I for eksempel standardkontrakten IKT 2010 sier pkt. 3.2.1, første avsnitt at leverandøren skal gjennomføre prosjektet på en «[…] fagmessig forsvarlig måte», og at han skal «[…] bruke materialer, teknikker og standarder av vanlig god kvalitet […]». I SSA-S finnes ingen tilsvarende bestemmelser, men det hersker liten tvil om at slike krav er en del av bakgrunnsretten for tilvirkningskontrakter. Også Unidroit har slike retningslinjer, art. 5.1.6:

> «Where the quality of performance is neither fixed by, nor determinable from, the contract a party is bound to render a performance of a quality that is reasonable and not less than average in the circumstances.»

Det er også klart at bransjenormen som skal legges til grunn, er den som gjelder ved kontraktsinngåelsen.[314] For programvareutvikling hvor den tekniske utviklingen skjer raskt, er denne regelen praktisk. For eksempel kan en anerkjent og vanlig måte å utvikle en sikkerhetsmekanisme på ved kontraktsinngåelsen, senere vise seg å ikke lenger tilfredsstille sikkerhetsnivået. Da kan ikke programvaren ansees for å være mangelfull.

Utfordringen er å fastslå hvilke av bransjens normer som kan legges til grunn ved mangelsvurderingen. I entreprise kan ulike standarder være så innarbeidede at i hvert fall deler av standardene fastslår sedvaner innen bransjen, men likevel må kontrakten henvise til standardene for at de skal gjelde.[315] Det finnes også visse standarder knyttet til programvareutvikling, som for eksempel NS-ISO/IEC 27001:2013, som gjelder krav ved utvikling av informasjonssikkerhet. Et annet eksempel er WCAG-standarden[316] med krav til utforming av brukergren-

312 Jf. Hagstrøm (2011) s. 170
313 Se Giverholt (2012) s. 245
314 Jf. Haaskjold (2013) s. 500
315 Jf. Sandvik (1966) s. 383
316 Se W3C (2012)

sesnitt for webbaserte systemer. Slike standarder er ikke vanlig å bruke i alle utviklingsprosjekter. På samme måte som i entreprisekontrakter, må slike standarder eksplisitt blir gjort til en del av kontrakten for bli gjeldende.

Krav til ytelsen som følger av offentligrettslig lovgivning kan være et annet grunnlag i den abstrakte mangelsvurderingen.[317] Når entreprenøren i for eksempel totalentreprise har spillerom med tanke på innholdet i ytelsen, vil bygningslovgivningen begrense valgfriheten.[318] Dermed er entreprenøren forpliktet til å holde seg innenfor lovgivningen. Dette kan også få betydning for leverandørens forpliktelse ved utvikling av programvare. Et praktisk tilfelle kan være knyttet til forskrift om universell utforming av it-løsninger.[319] Forskriften krever at «nettløsninger og automater» som «retter seg mot allmennheten i Norge» skal være universelt utformet, jfr. §§ 1 og 2. Denne forskriften setter klare krav til utformingen av brukergrensesnittet, og leverandøren må dermed være forpliktet til å levere programvare som oppfyller disse kravene. Tilsvarende problemstilling kan også gjelde for forskriften om behandling av personopplysninger,[320] som også setter krav til både kunde og leverandør i arbeidet med løsninger som skal inneholde informasjon som reguleres av forskriften. Når det gjelder behandling av personopplysninger stiller dette også krav til prosedyrer for begge parter. Derfor har for eksempel SSA-S tatt inn en egen bestemmelse som omhandler personopplysninger, jf. pkt. 9.2

Som jeg tidligere har vært inne på, er programvareutvikling en ung ingeniørdisiplin i rask utvikling. Derfor er det i liten grad utarbeidet *best practices* eller andre standarder som kan sies å etablere en norm i bransjen, se underkapittel 2.1.4. Hvilke normer som eventuelt er gjeldende i bransjen vil da kun komme frem ved «[…] fagkyndige konkrete vurderinger.»[321] Dermed blir det vanskelig å forutsi hva en slik norm innebærer. Konsekvensen av dette er at en smidig kontrakt, hvor spesifikasjonen ikke er uttømmende regulert, bør inneholde generelle krav til programvarens kvalitet. Hvis dette ikke er gjort, bør en likevel forutsette at det eksisterer en minimumsgrense for kvaliteten på det som utvikles. På noen områder innenfor programvareutviklingen har det etablert seg klare oppfatninger om hvordan strukturen i programkoden skal være, for eksempel ved bruk av objektorientert programmering. I tillegg bør programvaren ha en logisk oppdeling av funksjonsområder, slik at programkode for brukergrensesnittet bør være adskilt fra programkode som gjør beregninger og annen prosessering av data. En slik oppdeling kalles gjerne for *flerlagsstruktur*. For modellering av

317 Jf. Sandvik (1966) s. 384
318 Jf. Haaskjold (2013) s. 565 (petit)
319 Forskrift om universell utforming av IKT-løsninger (2013)
320 Personopplysningsforskriften (2000)
321 Jf. Krüger (1989) s. 221

relasjonsdatabaser[322] finnes det også metoder som bør anses som normer. For eksempel at en velfungerende og strukturert databasemodell forutsetter en viss grad av normaliserte tabeller[323] med data, og at sorteringsindeksene[324] gjenspeiler normaliseringen.

Det kan også være aktuelt å stille minstekrav til brukergrensesnittets estetikk. Et slik krav kan bygge på et tilsvarende krav i totalentreprise, hvor Barbo trekker frem en voldgiftsdom der estetikken fremsto med «[…] 'omtrentlig og lite gjennomarbeidet inntrykk' […]» og kunne ansees som en mangel.[325]

En utfordring når det ikke finnes etablerte standarder, er at eventuell konfliktløsning i slike prosjekter dreier seg mye om tekniske kvalitetsforhold som krever bransjeinnsikt.[326] Av den grunn kan en konfliktløsningsmodell med prosjektintegrert mekling, se underkapittel 3.6.5, være et godt utgangspunkt for å løse uenigheter om kvaliteten i programvaren. Det forutsetter imidlertid at mekleren(e) har tilstrekkelig teknisk kompetanse.

Hvilket kvalitetsnivå som kan forventes, kan også knyttes til prisen.[327] Spesielt i fastpriskontrakter kan kvaliteten bli satt under press hvis tiden og funksjonsomfanget er fast, jf. «The Iron Triangle» i underkapittel 1.4.3. Når vederlaget beregnes basert på regningsarbeid, kan kunden i større grad påvirke kvaliteten ved at leverandøren legger frem alternative løsninger som kunden kan ta stilling til. Da blir det opp til kunden å vurdere budsjettrammene med tanke på ønsket kvalitet. Det kan være fristende for kunden å velge de rimeligste alternativene for å spare penger, men på lengre sikt vil dette medføre såkalt *teknisk gjeld* i programvaren. Den tekniske gjelden består i at lav kvalitet på kodestruktur og arkitektur i systemet, medfører merarbeid på et senere tidspunkt når systemet skal videreutvikles og vedlikeholdes. En slik teknisk gjeld som kunden selv har pådratt seg, kan derfor ikke anses som en mangel som leverandøren har risikoen for. Men, selv om kunden velger et rimelig alternativ, er det ikke slik at leverandøren er fritatt for kravet om en fagmessig standard, som er hovedregelen. Det skal mye til for at man skal kunne si at hovedregelen er fraveket, og at kunden dermed har gitt avkall på en fagmessig standard.[328]

---

322 Relasjonsdatabaser ble utviklet på 1970-tallet, og er fortsatt den meste brukte strukturingsmetoden i databaser.

323 Normalisering av tabeller betyr å gruppere alle data i tabeller hvor informasjon som logisk sett hører sammen blir samlet.

324 Sorteringsindekser er helt nødvendig for at søkehastigheten i databasen skal være tilfredsstillende.

325 Se Barbo (1990) s. 59

326 Jf. Krüger (1989) s. 217

327 Jf. Sandvik (1966) s. 381

328 Se Krüger (1989) s. 214

3.7.5 Sanksjoner

Ved kontraktsbrudd kan det bli aktuelt med sanksjoner mot den parten som har risikoen for kontraktsbruddet. Det vil si at *årsaken* til oppfyllelsessvikten kan spores tilbake til parten. Dette kapittelet handler om sanksjoner knyttet til forsinkelse og mangler. Hvilke typer sanksjoner som er aktuelle, er avhengig av om det er en forsinkelse eller mangel. Tilbakehold av egen ytelse er mest praktisk ved forsinkelse, mens prisavslag er kun aktuelt når det er snakk om mangler. Andre sanksjoner som gjelder både forsinkelse og mangler er: Krav om oppfyllelse/utbedring, heving og erstatning. Krav om oppfyllelse i henhold til kontrakt er en naturlig konsekvens av at partene skal oppfylle sine plikter etter kontrakten.[329] Det er gjerne ved forsinkelse at dette kravet omtales som et krav om oppfyllelse, mens det ved mangler omtales det som retting.[330] Jeg velger å kalle denne sanksjonen for *utbedring*. Utbedringen kan gjennomføres ved dom på naturoppfyllelse.[331] Jeg skal ikke gå inn på alle sanksjonstyper og kombinasjoner av disse, men se nærmere på noen aspekter som gjør seg gjeldende i smidige kontrakter.

3.7.5.1 Dagbøter eller ikke?

Kunden skal gjennomføre akseptansetesting av en delleveranse, jf. avsnitt 3.6.1.1, og underveis i testingen vil kunden varsle leverandøren hvis feil og mangler som avdekkes, senest ved avslutningen av testingen. Hvis leveransen ikke har feil eller mangler som kan ansees som kontraktsbrudd, må kunden godkjenne leveransen. Hvis kunden ikke godkjenner leveransen, har leverandøren plikt til å utbedre manglene. Hvis det er avtalt muligheter for tilleggsfrist, vil leverandøren som regel benytte seg av denne med tanke på utbedringen. Hvis leverandøren ikke klarer å utbedre innen avtalte frister, er leveransen forsinket. For kunden vil det da være praktisk å kreve videre utbedring og/eller erstatning. Erstatning skal i utgangspunktet dekke eventuelle tap kunden har som følge av forsinkelsen. Det er som regel vanskelig å beregne hvor store tap kunden har lidt, og i noen tilfeller vil ikke kundens nytte av systemet kunne kobles til en økonomisk gevinst.[332] Derfor avtales det ofte *dagbøter* ved forsinkelser, som er både er en form for erstatning og et pressmiddel overfor leverandøren.

Som jeg var inne på ved drøftelsene av prismodeller og insentivsystemer for smidige kontrakter, kan det diskuteres om insentivsystemer skal brukes eller ikke. Min mening er at man bør unngå insentivsystemer, både negative og positive, for å opprettholde enkeltheten i kontrakten. Derfor bør man heller ikke bruke økonomisk straff som dagbøter. Det kan likevel være situasjoner hvor for eksempel kundens nytte av programvaren er helt avhengig av levering innen

329 Jf. Hagstrøm (2011) s. 377
330 Jf. Hov (2007) s. 130
331 l.c. og Rt. 1972 s. 449 (s. 455 om reparasjonsplikten)
332 Jf. Torvund (1997) s. 176

visse tidsrammer, og at dagbøter dermed kan ha sin funksjon for å legge ekstra press på leverandøren.

SSA-S har bestemmelser om at dagbøter starter å løpe automatisk ved forsinkelser, jf. pkt. 11.5.2. Ofte kombineres dagbøtene med en hevingsrett etter en viss tid, jf. SSA-S, pkt. 11.5.4. I SSA-S er størrelsen på dagboten 0,15 prosent av delleveransens relative verdi sammenliknet med estimert totalkostnad, jf. pkt. 11.5.2, tredje avsnitt. En slik beregning av dagbøter forutsetter at det er utarbeidet et ganske sikkert kostnadsoverslag, som til en viss grad er bindende for partene. I en smidig kontrakt, slik jeg har skissert den, er det etter mitt syn vanskelig å knytte dagbøter til en budsjettramme. Spesielt når kunden underveis aktivt er med å prioritere og endre hva som skal utvikles. Det tette samarbeidet gjør også at det kan være samvirkende årsaker til forsinkelsen, og at partene derfor bør dele risikoen. Når utviklingen skjer ved regningsarbeid skal som nevnt kunden betale for utbedringen i forsinkelsesperioden.

På bakgrunn av disse forholdene mener jeg en alternativ løsning er at leverandørens timepris for utbedringsarbeid reduseres i forsinkelsesperioden. Da får både leverandøren press på seg for å løse problemene, og kunden får en viss normalerstatning for kontraktsbruddet. Størrelsen på reduksjonen i timeprisen kan bero på en konkret vurdering, eller avtales ved kontraktsinngåelsen, slik at det blir en rettsteknisk forenkling. Spørsmålet blir hvor lenge forsinkelsesperioden med redusert timepris skal være, og hva som skal skje når perioden er over. I både SSA-S, pkt. 11.5.2 og i PS2000-kontraktene, pkt. 6.1.1.1 er dagbotperioden hundre kalenderdager.

Hvis levering uten mangler ikke har funnet sted etter utløpet av perioden, er det vanlig at kunden kan heve avtalen og kreve erstatning.[333] Prisavslag er som regel knyttet til retting av mangler etter levering, men for eksempel i SSA-S, pkt. 11.5.3 kan kunden også kreve prisavslag ved forsinkelse. Det gjelder hvis leverandøren, tross gjentatte utbedringsforsøk, ikke makter å løse problemene.

3.7.5.2 Utbedring og prisavslag

Når det gjelder mangler etter levering, ønsker som regel begge parter at manglene blir utbedret. Når kontrakten har regningsarbeid som prismodell, er det i utgangspunktet kunden som betaler for utbedringen. Kunden ville uansett ha måttet betale for det manglende arbeidet som førte til oppfyllelsessvikten.[334] Imidlertid kan ikke leverandøren få dekket ekstra kostnader som skyldes at mangelen utbedres etter levering. Det vil si at kunden ikke skal betale mer for utbedringen, enn det han måtte betalt hvis arbeidet ble utført rett første gangen.[335] En slik regel er praktisk, fordi det ofte kan være slik at det vil medføre

333 Jf. Torvund (1997) s. 183

334 Jf. Sandvik (1966) s. 186

335 Se Simonsen (1999) s. 351

betydelig merarbeid å utbedre en mangel når systemet er levert, enn arbeidet for å unngå mangelen i første omgang.

Hvis leverandøren ikke retter mangelen eller ikke klarer å rette den, kan økonomisk kompensasjon være et alternativ. Dette kan være et prisavslag, som kan beregnes på flere måter. I utgangspunktet er formålet med prisavslag at det skal «[…] sikre balansen mellom ytelsene.»[336] Beregningen gjøres da forholdsmessig ved å se på avtalt pris (som antas å være markedsverdi) sammenliknet med verdien i mangelfull tilstand. Denne beregningsmåten er lite praktisk for spesialtilpasset programvare. Det er generelt vanskelig, og kanskje umulig, å fastsette en objektiv markedspris for spesialtilvirkede løsninger.[337] Andre beregningsmåter for et prisavslag kan være erstatning for verdiminuset eller for utbedringskostnader. Begge deler er erstatning som må baseres på et ansvarsgrunnlag. Erstatningsgrunnlaget som kan være aktuelt i denne sammenheng er et eventuelt ulovfestet kontrollansvar. Det er imidlertid usikkert om det gjelder et slikt kontrollansvar for andre ytelser enn genusytelser, men Hagstrøm har argumentert for at det også gjelder kontrollansvar for andre ytelser.[338] På grunn av uklarheten bør derfor en smidig kontrakt inneholde klare bestemmelser om prisavslag og beregningen av dette.

Prisavslag beregnet etter erstatning for verdiminuset, vil støte på de samme utfordringene som forholdsmessig beregning. Det er fortsatt en markedsverdi som skal legges til grunn. For å fastsette et prisavslag når det gjelder spesialtilvirket programvare, kan beregningen ta utgangspunkt i hva det ville kostet å utbedre manglene.[339] Denne beregningsmåten er lagt til grunn i håndverkertjenesteloven[340] § 25 (2), første punktum: «Prisavslaget settes til det det ville koste forbrukeren å få mangelen rettet […]». I en smidig kontrakt hvor tilvirkningen skjer ved regningsarbeid, må beregningsmåten imidlertid justeres. Kunden måtte uansett ha betalt for det manglende arbeidet tidligere i utviklingsprosessen. Derfor må det gjøres fradrag i utbedringskostnadene for de kostnadene leverandøren ville hatt ved å utført arbeidet i første omgang for å unngå mangelen. Konsekvensene av dette blir i mange tilfeller at størrelsen på prisavslaget blir ubetydelig. Men, som nevnt, kan utbedringskostnadene bli en del større når manglene må rettes etter at programvaren er utviklet, og da vil prisavslaget likevel ha en betydning. Hvis mangelen ikke lar seg utbedre, må prisavslaget fastsettes skjønnsmessig.[341]

Både PS2000-kontraktene, pkt. 6.1.2.2, og SSA-S, pkt. 11.5.3, beregner et forholdsmessig avslag. Tilsvarende bestemmelser har NS 8407, men med et tillegg

336 Jf. Rt. 2000 s. 199 (s. 206)
337 Jf. Simonsen (1999) s. 365
338 Jf. Hagstrøm (2011) s. 523
339 ibid. s. 418 og Simonsen (1999) s. 366
340 Håndverkertjenesteloven (1989)
341 Jf. Hagstrøm (2011) s. 418

om at: «Prisavslaget skal minst svare til den besparelsen totalentreprenøren har oppnådd som følge av at utførelsen ikke er kontraktsmessig», jf. pkt. 42.3.4, 2. avsnitt.

I en smidig kontrakt vil nok prisavslag generelt sett være lite praktisk. Mangler som avdekkes etter levert delleveranse, vil bli utbedret av leverandøren i en senere sprint. Det som imidlertid er mer praktisk, er prisavslag for eventuelle mangler som oppdages etter at prosjektet er avsluttet.

3.7.5.3 Heving

Heving er en sanksjon som kan ramme partene hardt. Derfor er det krav til at forsinkelsen eller mangelen skal være vesentlig. Som tidligere nevnt bør partene før situasjonen er kommet så langt aktivere konfliktløsningsmetoden prosjektintegrert mekling. I tillegg vil partene i det løpende samarbeidet avdekke problemer før man kommer så langt som at heving på grunn av vesentlige forsinkelser eller mangler er aktuelt. Begge parter har også mulighet for løpende oppsigelse av kontrakten. Selv om en slik oppsigelse medfører et økonomisk oppgjør, kan det likevel være en rimeligere og mindre dramatisk avslutning av samarbeidet enn å heve kontrakten. Likevel er heving en siste utvei i tilfeller hvor leverandøren ikke klarer å utbedre mangler i løpet av en forsinkelsesperiode, eller vesentlige mangler etter levering. I en smidig kontrakt med løpende delleveranser som kan settes i produksjon bør det kun være aktuelt med heving som virker fremover (ex nunc). Det forutsetter imidlertid at kunden overtar rettigheter til endring og videreutvikling av programvare som allerede er levert. For det første vil kunden da kunne benytte allerede levert programvare, og andre leverandører kan videreføre utviklingen. For det andre vil ikke heving ex nunc ramme leverandøren så hardt. Uansett vil det komme på tale med et økonomisk oppgjør i tillegg, slik at kundens situasjon kan restitueres.

3.7.5.4 Erstatning

Som nevnt er utgangspunktet ved erstatning at det må foreligge et ansvarsgrunnlag, og det er uklart om det er et ulovfestet kontrollansvar for utviklingskontrakter. Når kontraktene regulerer spørsmålene om erstatning, blir det av mindre interesse å ta stilling til kontrollansvaret. I tillegg vil det være i partenes interesse å regulere erstatningsmuligheten for å ha kontroll over den økonomiske risikoen. I SSA-S kan kunden «[…] kreve erstattet ethvert direkte tap som følger av forsinkelse, mangel eller annet mislighold», jf. pkt. 11.5.5, første avsnitt. Dette forutsetter at årsaken til kontraktsbruddet ikke ligger hos leverandøren, jf. risikofordelingen. Eventuelle dagbøter skal gå til fradrag i erstatningen, jf. pkt. 11.5.5, andre avsnitt. Ansvaret i SSA-S ligger etter min mening tett opp til kontrollansvaret. I PS2000-kontraktene kan kunden kreve erstatning for sannsynliggjorte tap som følge av forsinkelse etter utløpet av dagbotperioden, jf. pkt.

6.1.1.2. Når det gjelder vesentlige mangler finnes tilsvarende bestemmelse i pkt. 6.1.2.4.

I avsnitt 3.7.5.2 drøftet jeg prisavslag, som også er en form for erstatning. Størrelsen på prisavslaget ved bruk av regningsarbeid kan som nevnt bli ubetydelig. Et objektivt ansvar for erstatning som dekker direkte tap, kan rekke lenger enn beregningsreglene for prisavslag. Derfor kan det, i en smidig kontrakt, være hensiktsmessig å ta inn bestemmelser om erstatning for direkte tap med et ansvarsgrunnlag som ligger tett opp til kontrollansvaret.

I avsnitt 3.7.1.4 trakk jeg frem erstatning som sanksjon i forbindelse med partenes subjektive forhold. Partenes subjektive forhold vil først og fremst gi seg utslag på risikofordelingen mellom partene, som igjen vil være styrende for andre sanksjoner. Erstatningen kan likevel være aktuell for eksempel i forbindelse med et økonomisk oppgjør ved oppsigelse, hvis en av partene har opptrådt uansvarlig med tanke på kontraktens forpliktelser. I kontrakter hvor partene jobber tett sammen, kan det være vanskelig å avgjøre årsaksforholdene. Det kan være flere samvirkende årsaker som har ført til tapet. For eksempel i en lagmannsrettsdom ble erstatningens opprinnelige størrelse halvert, fordi begge partene kunne klandres.[342]

Erstatningsansvaret rekker i utgangspunktet langt, og faren for store økonomiske konsekvenser for begge parter gjør at ansvaret som regel begrenses i kontraktene.[343] En slik ansvarsbegrensning bør også legges inn i en smidig kontrakt, slik at partene har en viss økonomisk forutsigbarhet. Ansvarsbegrensningene kan for eksempel knyttes til beløpsgrenser eller hvilke tapsposter som dekkes.[344]

Når det gjelder ansvarsbegrensninger for grov skyld, er det mer tvilsomt hvor langt slike bestemmelser rekker. I Rt. 1994 s. 626 gjaldt det en standardkontrakt som hadde en ansvarsbegrensning for grov skyld. Høyesterett valgte å ikke sette bestemmelsen til side, og la vekt på at det var en fremforhandlet kontrakt («agreed documents»).[345] I andre kontrakter er det ikke usannsynlig at slike ansvarsbegrensninger blir sensurert på bakgrunn av avtalelovens[346] § 36.[347] Verken SSA-S, jf. pkt. 11.5.6, eller PS2000-kontraktene, jf. pkt. 10.4, siste avsnitt, har ansvarsbegrensninger ved grov skyld.

3.7.6 Garanti og vedlikehold

Som jeg har vært inne på vil sanksjoner basert på mangler kun være aktuelt etter at en delleveranse er levert. Kunden skal i akseptansetestperioden undersøke leveransen med tanke på mangler. Etter at kunden har akseptert leveransen kan

342 Jf. LB-2004-102939 pkt. 5.3
343 Jf. Føyen (2006) s. 468
344 Se Torvund (1997) s. 236
345 Se Rt. 1994 s. 626 (s. 630)
346 Avtaleloven (1918)
347 Jf. Simonsen (1999) s. 380

det være snakk om reklamasjon for typisk skjulte mangler. For at leverandøren skal ha forutsigbarhet med tanke på eventuelle reklamasjoner er det vanlig at standardkontrakter legger opp til en garantiperiode på 6-12 måneder. Dermed kan leverandøren planlegge for tilgjengelige ressurser for retting av mangler som kunden gjør gjeldende ved reklamasjon. Spørsmålet er om en smidig kontrakt skal inneholde bestemmelser om garanti.

Ved bruk av smidige metoder med hyppige leveranser, rettes manglene fortløpende. Også i andre tilvirkningsprosjekter, som for eksempel fabrikasjon, følger kunden nøye med på fremdriften og kontrollerer arbeidet underveis. Mangler og endringer blir forløpende tatt hånd om. Denne løpende rettingen vil som Mestad påpeker «[…] sørgje for at det som kunne ha blitt mangelsspørsmål alt er løyst eller i ferd med å bli løyst før levering.»[348]

Hvis leverandøren skal forplikte seg i en garantiperiode har dette en kostnad som enten må prises særskilt, eller som påslag ved regningsarbeid. Et annet moment er at hvis leverandøren likevel ikke retter, vil trolig resultatet bli et økonomisk oppgjør. Da kan garanti i større grad sees på som et spørsmål om økonomi, mer enn en praktisk ordning for å rette skjulte mangler. I tillegg vil ofte et vedlikeholdsopplegg følge etter prosjektet avslutning.

Med disse forholdene tatt i betraktning, mener jeg en smidig kontrakt bør legge opp til flere sprinter enn det som strengt tatt er nødvendig for å utvikle ønsket programvare. Disse «ekstra» sprintene kan imidlertid være nedskalert med tanke på at arbeidsomfanget er mindre. Disse sprintene kan da brukes til utbedring av eventuelle skjulte mangler, og ikke minst forbedringer som kunden ønsker. Det vil alltid oppstå nye behov og nye muligheter etter at alle delleveransene er satt i produksjon. Utnyttelse av disse mulighetene vil øke potensialet for gevinstrealisering. I tillegg kan leverandøren få mulighet til større inntjening hvis kunden er fornøyd med leverandørens ytelser.

348 Se Mestad (1999) s. 291

4 Oppsummering

Det er mange it-prosjekter som ikke lykkes. Årsakene er ofte at prosjektene har hatt store tids- og kostnadsoverskridelser, eller at systemet ikke kunne benyttes slik som kunden har forutsatt. Hvilke kriterier som ligger til grunn for at et prosjekt skal bli ansett som vellykket, vil variere fra prosjekt til prosjekt.

Et viktig kriterium som alltid vil følge it-prosjekter, er at anskaffelsen skal gi nytte og verdi for brukerne, organisasjonen og andre interessenter. Jeg har trukket frem en rekke særtrekk ved programvareutvikling, og vist hvordan disse skaper stor *usikkerhet* knyttet til hva som skal utvikles og hvordan det skal gjøres. Denne usikkerheten gjør at det ved kontraktsinngåelsen er vanskelig å spesifisere programvarens egenskaper.

Smidige utviklingsmetoder tar disse utfordringene inn over seg og legger opp til at programvare først skal spesifiseres i detalj underveis i utviklingsprosessen. Gjennom detaljspesifisering, evaluering og læring underveis kan leverandøren utvikle programvare som er bedre tilpasset kundens behov og dermed gir større nytteverdi.

Jeg har vist hvordan de tradisjonelle kontraktene ikke er tilpasset smidigtankegangen. Disse kontraktene forutsetter at det som skal utvikles er endelig og uttømmende spesifisert alt ved kontraktsinngåelsen. Når disse kontraktene likevel brukes i prosjekter som benytter smidig programvareutvikling, øker sjansen for at prosjektet mislykkes. Derfor er det behov for kontrakter tilpasset smidig programvareutvikling. Med utgangspunkt i disse forholdene har det overordnede spørsmålet i denne oppgaven vært: *Hvilke kontraktsmekanismer kan bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes?* Jeg har forsøkt å besvare dette spørsmålet, og dermed bidra i arbeidet med å utforme kontrakter egnet for smidig programvareutvikling.

Et sentralt utgangspunkt for kontraktsutformingen er spesifikasjonen av programvaren som skal utvikles. Derfor var utgangspunktet for drøftelsen av kontraktsmekanismene følgende problemstilling:

Hvordan kan alternative måter å spesifisere programvaren på bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes? Og hvordan skal dette reguleres i kontrakten med tanke på risikofordeling og kontraktsbrudd?

Jeg har konkludert med at et *effektkart* er egnet som spesifikasjon av et it-system når smidige metoder skal benyttes. Effektkartet bygger på kundens forventninger og ønsker om realisering av gevinster. Denne måten å spesifisere programvaren på stiller store krav til begge parter underveis i prosjektet, og gir leverandøren stor valgfrihet med tanke på hvordan funksjonene i systemet skal

utformes. Det tette samarbeidet basert på tillit må gjenspeiles i øvrige kontraktsmekanismer. Først og fremst har jeg skissert hvordan utviklingsprosessen og prosjektstyringen kan reguleres i kontrakten. Denne reguleringen, sammen med prismodellen, får betydning for risikofordelingen, og dermed hvordan kontraktsbrudd skal håndteres.

Den proaktive jussens tilnærming til kontraktsutforming har vært et bakteppe i alle drøftelsene. Jeg har latt meg inspirere av denne tilnærmingen når det gjelder utformingen av en enkel kontrakt, visuell presentasjon av spesifikasjonen og utviklingsprosessen, tidlig konfliktløsning samt risikofordeling. Også det smidige manifestet har spilt en viktig rolle i drøftelsene av kontraktsutformingen. I tillegg har jeg forsøkt å balansere partenes behov for både fleksibilitet og forutsigbarhet med tanke på kostnader og forventet resultat.

På bakgrunn av drøftelsene i denne oppgaven mener jeg en smidig kontrakt bør ta innover seg følgende elementer:

- Utgangspunktet bør være kundens *business case* der kunden klargjør hvorfor prosjektet skal etableres, hvem som er berørt og hvilke gevinster skal oppnås.
- Kunden utvikler en plan for hvordan gevinstene skal realiseres – både underveis i prosjektet og etter avsluttet prosjekt: *Gevinstrealiseringsplan.*
- *Effektmålene* trekkes ut fra gevinstrealiseringsplanen og er grunnlaget for utarbeidelse av et *effektkart.*
- Effektkartet utarbeides ved å trekke inn *brukergrupper*, samt spesifikasjon av overordnet funksjonalitet i form av *epos* og *brukerhistorier.*
- Dette effektkartet legges til grunn ved kontraktsinngåelsen som *spesifikasjon av kontraktsgjenstanden*.
- *Utviklingsprosessen* beskrives detaljert i kontrakten med tilhørende visuell fremstilling, slik at partenes funksjonsfordeling kommer tydelig frem.
- Prosjektstyringen gjennomføres av en *gruppe for felles prosjektstyring*, som både tar visse beslutninger og løser konflikter.
- Fremdriftsplan utarbeides i fellesskap med krav til *delleveranser.*
- Endringshåndtering *uten preklusive frister.*
- *Prosjektintegrert mekling* er viktig for å forebygge og raskt løse konflikter.
- Tydeliggjøring av en skjerpet *gjensidig lojalitets- og varslingsplikt.*
- *Regningsarbeid* som prismodell – ingen positive eller negative insentiver.
- *Løpende oppsigelsesmuligheter* for begge parter.
- *Reduksjon av timepris* som et alternativ til dagbot.
- *Vedlikeholdsperiode* istedenfor garanti- eller feilrettingsperiode.

Gjensidig tillit mellom partene er en forutsetning for å bruke smidige kontrakter. Det kreves også høy grad av modenhet for den smidige tankegangen. Jeg er

usikker på om markedet i dag har den graden av modenhet som skal til for å bruke smidige kontrakter. Samtidig tror jeg dette vil endre seg. Det er ingenting som tyder på at pendelen svinger tilbake til de tradisjonelle utviklingsmetodene og fossefallsmetoden. Stadig flere it-prosjekter tar i bruk smidig programvareutvikling, og markedet trenger funksjonelle, smidige kontrakter om disse prosjektene skal lykkes.

5 Vedlegg

5.1 Prinsippene bak det smidige manifestet

Prinsippene bak Det smidige manifestet

Vi følger disse prinsippene:

Vår høyeste prioritet er å tilfredsstille kunden
gjennom tidlige og kontinuerlige leveranser
av programvare som har verdi.

Ønsk endringer i krav velkommen, selv sent i utviklingen.
Smidige prosesser bruker endringer til
å skape konkurransefortrinn for kunden.

Lever fungerende programvare hyppig,
med et par ukers til et par måneders mellomrom.
Jo oftere, desto bedre.

Forretningssiden og utviklerne må arbeide sammen
daglig gjennom hele prosjektet.

Bygg prosjektet rundt motiverte personer.
Gi dem miljøet og støtten de trenger,
og stol på at de får jobben gjort.

Den mest effektive måten å formidle informasjon
inn til og innad i et utviklingsteam,
er å snakke ansikt til ansikt.

Fungerende programvare er det primære målet på fremdrift.

Smidige metoder fremmer bærekraftig programvareutvikling.
Sponsorene, utviklerne og brukerne bør kunne
opprettholde et jevnt tempo hele tiden.

Kontinuerlig fokus på fremragende teknisk kvalitet
og godt design fremmer smidighet.

Enkelhet – kunsten å maksimere mengden arbeid
som ikke blir gjort – er essensielt.

De beste arkitekturer, krav og design
vokser frem fra selvstyrte team.

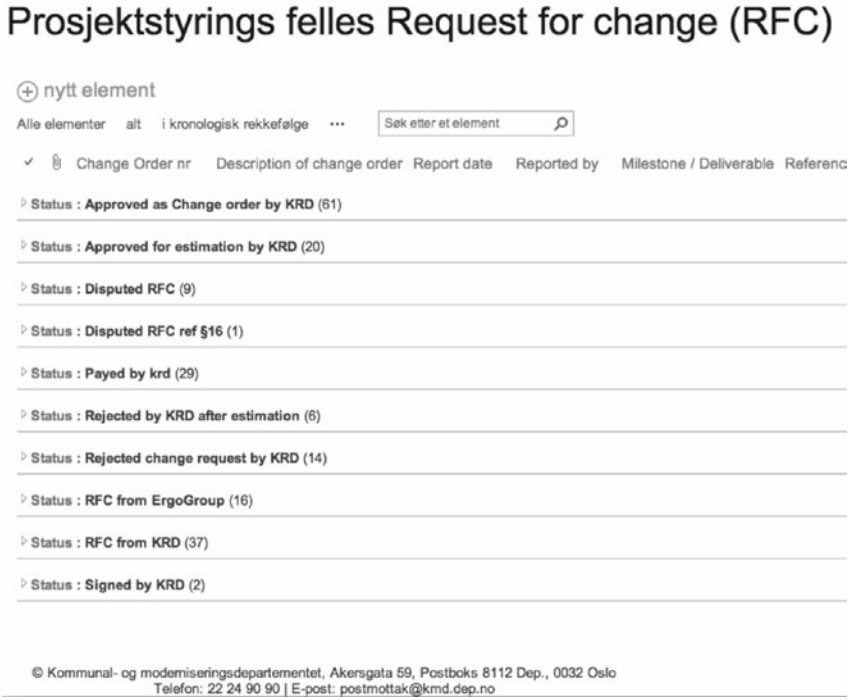
Med jevne mellomrom reflekterer teamet over
hvordan det kan bli mer effektivt og
så justerer det adferden sin deretter.

5.2 E-valgforsøket 2011

Appendix 7 Total price and pricing provisions	All costs in NOK excluded VAT/MVA
Date	15.12.2009
Company Name	ErgoGroup AS
Exchange rate used	
Currency used	NOK

Totals (Sum from tables 1 - 7 below)	Total contract value ex. Options		Total value of options		Total Cost used for price comparison	
1: Total cost Optional services, Personnel cost pr hour		NA	kr	2 300 000	kr	2 300 000
2: Total Cost Development / customisation of Use Cases	kr	7 514 500	NA		kr	7 514 500
3: Total Cost Development / customisation of Options		NA	kr	2 002 450	kr	1 312 178
4: Total Cost Software Components	kr	18 675 750	kr	611 250	kr	19 164 750
5: Total Cost HW 2011		NA		NA	kr	420 360
6: Sum Optional services for Implementation (2011) and optional project cost		NA	kr	1 823 959	kr	1 414 006
7: Total Cost Other Contractor project costs until September 2011	kr	2 155 500		NA	kr	2 155 500
Grand Total	kr	28 345 750	kr	6 737 659	kr	34 281 293

Figur 5.1: Kontraktssummer i e-valg2011-prosjektet. (Kilde: Kommunal- og moderniseringsdepartementet)



Figur 5.2: Antall endringsordrer i e-valg2011-prosjektet. (Kilde: Kommunal- og moderniseringsdepartementet)

5.3 Eksempel på et effektmål og effektkart

Effektmål: Redusere bedriftens rentetap på utestående fordringer med 20 %

1. **Brukergruppe**: Ordremottaker
   - 1.1. **Epos**: Som ordremottaker ønsker jeg å ha tilgang til all informasjon om en kunde/bestiller
     - 1.1.1. **Brukerhistorie**: Som ordremottaker ønsker jeg å søke etter en kunde og vise all informasjon om en kunde slik at jeg slipper å bla frem og tilbake mellom flere skjermbilder.
     - 1.1.2. **Brukerhistorie**: Som ordremottaker ønsker jeg å vente maksimum 3 sekunder på søk etter kundeinformasjon slik at jeg ikke bruker unødvendig tid på venting.
   - 1.2. **Epos**: Som ordremottaker ønsker jeg å ha en raskere måte å legge inn en ordre på
     - 1.2.1. **Målepunkt**: 15 min gjennomsnittlig behandlingstid
     - 1.2.2. **Brukerhistorie**: Som ordremottaker ønsker jeg at en ordre er ferdig utfylt med eksisterende kundedata slik at jeg sparer tid på manuell utfylling.
     - 1.2.3. **Brukerhistorie**: Som ordremottaker ønsker jeg å duplisere tidligere ordrer på en kunde slik at jeg sparer tid på manuell utfylling av eksisterende info.
     - 1.2.4. **Brukerhistorie**: Som ordremottaker ønsker jeg å søke etter ordrer og kunder i samme søk slik at det går raskere å finne riktig kunde.
2. **Brukergruppe**: Kunde/bestiller
   - 2.1. **Epos**: Som kunde/bestiller ønsker jeg legge inn en ordre selv
     - 2.1.1. **Målepunkt**: Minst 20 % av bestillingene
     - 2.1.2. **Brukerhistorie**: Som kunde/bestiller ønsker jeg å legge inn ny ordre
     - 2.1.3. **Brukerhistorie**: Som kunde/bestiller ønsker jeg å velge om jeg vil ha e-faktura, papir- eller pdf-faktura
   - 2.2. **Epos**: Som kunde/bestiller ønsker jeg administrere mine egne ordrer og fakturaer
     - 2.2.1. **Målepunkt**: Redusere behandlingstiden med 50 %
     - 2.2.2. **Brukerhistorie**: Som kunde/bestiller ønsker jeg at ingen uvedkommende får tilgang til mine kundeopplysninger, ordrer og fakturaer
     - 2.2.3. **Brukerhistorie**: Som kunde/bestiller ønsker jeg å ha oversikt over status mine ordre, avbestille, endre eller lage ny ordre

3. **Brukergruppe**: Økonomiavdeling
    - 3.1. **Epos**: Som økonomiavdeling ønsker jeg fortløpende å godkjenne fakturaer
        - 3.1.1. **Målepunkt**: Redusere behandlingstid med 50 %
        - 3.1.2. **Brukerhistorie**: Som økonomiavdeling ønsker jeg å få automatisk varsel når fakturaene er klar til godkjenning.
        - 3.1.3. **Brukerhistorie:** Som økonomiavdeling ønsker jeg å endre på en faktura før godkjenning.

6 Figurliste

Kilder

Litteratur

Adzic (2012) : Gojko Adzic. *Impact Mapping: Making av big impact with software products and projects*. United Kingdom, 2012.

Ambler (2012) : Scott W. Ambler og Mark Lines. *Disciplined agile delivery – A Practitioner´s guide to agile software delivery in the enterprise*. Upper Saddle River, N.J., 2012.

Atkinson (1999) : Roger Atkinson. *Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria*. I: International Journal of Project Management. Vol. 17 (1999). s. 337-342.

Atkinson (2011) : Susan Atkinson og Gabrielle Benefield. *The Curse of the Change Control Mechanism*. I: Computers & Law Magazine of SCL. Vol. 22 (2011). s. 1-6.

Atkinson (2013) : Susan Atkinson og Gabrielle Benefield. *Software Development: Why the Traditional Contract Model Is Not Fit for Purpose*. System Sciences (HICSS), 2013 46th Hawaii International Conference on, 2013. s. 4842-4851.

Barbo (1990) : Jan Einar Barbo. *Totalentreprise – særlig om entreprenørens prosjekteringsrisiko*. Oslo, 1990.

Beck (2004) : Kent Beck og Cynthia Andres. *Extreme programming explained: embrace change*. Boston, 2004.

Berger-Walliser (2012) : Gerlinde Berger-Walliser. *The Past and Future of Proactive Law: An Overview of the Development of the Proactive Law Movement*. I: Proactive Law in a Business Environment. Gerlinde Berger-Walliser og Kim Østergaard (red.). Danmark, 2012.

Bergsaker (2010) : Olav Bergsaker. *Samspillkontrakter – noen refleksjoner om de rettslige rammene*. I: På rett grunn – Festskrift til Norsk Forening for Bygge- og Entrepriserett. Jan Einar Barbo og Lasse Simonsen (red.). Oslo, 2010. s. 170-185.

Boehm (1988) : B. W. Boehm. *A spiral model of software development and enhancement*. I: Computer. Vol. 21 (1988). s. 61-72.

Brevik (2013) : Eivind Brevik og Tor-Morten Grønli. *The Scrum Illusion: Use of Software Development Methods in the Norwegian IT-industry.* NOKOBIT 2013 – Norsk konferanse for organisasjoners bruk av informasjonsteknologi, 2013. s. 15-28. http://tapironline.no/fil/vis/1293. [Sitert 25.01.2014].

Brooks Jr. (1995) : Frederick P. Brooks Jr. *The mythical man-month: essays on software engineering*. Reading, Mass., 1995.

Buch (2007) : Anders Vestergaard Buch. *Entrepriseretlige mangler – kravene til entreprenørens ytelse*. København, 2007.

Cohn (2004) : Mike Cohn. *User stories applied: for agile software development*. Boston, Mass., 2004.

Cohn (2005) : Mike Cohn. *Agile estimating and planning*. Upper Saddle River, N.J., 2005.

Cohn (2010) : Mike Cohn. *Succeeding with agile: software development using Scrum*. Upper Saddle River, N.J., 2010.

Dragsted (2012) : Nicolai Dragsted. *It-projektkontrakter baseret på gevinstrealisering del 1*. I: Lov&Data nr. 109 (2012). s. 13-17.

Eide (2008) : Erling Eide og Endre Stavang. *Rettsøkonomi*. Oslo, 2008.

Føyen (2006) : Arve Føyen [et al.]. *Kontrakter for utvikling av programvare*. Oslo, 2006.

Gilb (2006) : Tom Gilb. *No Cure No Pay: How to Contract for Software Services* (2006). http://www.gilb.com/dl498. [Sitert 03.02.2014].

Gilb (2005) : Tom Gilb og Lindsey Brodie. *Competitive engineering: a handbook for systems engineering, requirements engineering, and software engineering using Planguage*. Amsterdam, 2005.

Gilb (2010) : Tom Gilb og Lindsey Brodie. *Values for Value*. I: Agile Record nr. 4 (2010). s. 14-22. http://www.gilb.com/tiki-download_file.php?fileId=448. [Sitert 03.02.2014].

Giverholt (2012) : Heikki Giverholt [et al.]. *NS 8407: alminnelige kontraktsbestemmelser for totalentrepriser med kommentarer*. Oslo, 2012.

Gleick (1992) : James Gleick. *Chasing Bugs in the Electronic Village*. I: The New York Times 1992. http://www.nytimes.com/1992/06/14/magazine/chasing-bugs-in-the-electronic-village.html. [Sitert 25.03.2014].

Hagstrøm (2011) : Viggo Hagstrøm. *Obligasjonsrett*. 2. utg. Oslo, 2011.

Hellang (2012) : Øyvind Hellang [et al.]. *Metoder og teknikker for gevinstrealisering*. I: Gevinstrealisering og offentlige IKT-investeringer. Leif Skiftenes Flak (red.). Oslo, 2012.

Henschel (2012) : René Franz Henschel. *Iterative Contracts as Proactive Law Instruments*. I: Proactive Law in a Business Environment. Gerlinde Berger-Walliser og Kim Østergaard (red.). Danmark, 2012.

Holsøe (2013) : Tom Holsøe og Mia Thulstrup Gedbjerg. *K03: Ny dansk standardkontrakt for længrevarende it-projekt baserer på en agil metode*. I: Lov&Data nr. 113 (2013).

Hov (2007) : Jo Hov. *Avtalebrudd og partsskifte: Kontraktsrett II*. 3. utg. Oslo, 2007.

Haapio (2013) : Helena Haapio. *Next Generation Contracts: A Paradigm Shift*. Helsinki, 2013.

Haaskjold (2013) : Erlend Haaskjold. *Kontraktsforpliktelser*. Oslo, 2013.

Karlsen (2013) : Jan Terje Karlsen. *Prosjektledelse – fra initiering til gevinstrealisering*. 3. utg. Oslo, 2013.

Knag (2010) : Alf Johan Knag. *Hvordan skrive kontrakt? – en sann historie om ønsketenkning, ukyndighet og annen konfliktskapende virksomhet*. I: På rett grunn – Festskrift til Norsk Forening for Bygge- og Entrepriserett. Jan Einar Barbo og Lasse Simonsen (red.). Oslo, 2010. s. 85-110.

Kolrud (2009) : Helge Jakob Kolrud. *Hvilke krav bør stilles til en god endringsbestemmelse?* I: Festskrift til det Danske Selskab for Byggeret. København, 2009. s. 151-162.

Krokeide (1977) : Kjetil Krokeide. *Forutsetningslæren og misligholdsbegrepet – særlig i langsiktige kontraktsforhold*. I: Tidsskrift for rettsvitenskap (1977). s. 569-649.

Krüger (1989) : Kai Krüger. *Norsk kontraktsrett*. Bergen, 1989.

Krüger (2010) : Kai Krüger. *Tolkning av kontrakter som er tildelt ved anbudskonkurranse*. I: På rett grunn – Festskrift til Norsk Forening for Bygge- og Entrepriserett. Jan Einar Barbo og Lasse Simonsen (red.). Oslo, 2010. s. 111-129.

Kaasen (1994) : Knut Kaasen. *Partnering: Keiserens nye klær?* I: MarIus nr. 209 (1994). s. 31-40.

Kaasen (2005) : Knut Kaasen. *"Dynamisk kontraktsrett" – et fruktbart grep?* I: Tidsskrift for rettsvitenskap. Vol. 3 (2005). s. 237-263.

Kaasen (2006) : Knut Kaasen. *Petroleumskontrakter: med kommentarer til NF 05 og NTK 05*. Oslo, 2006.

Kaasen (2009) : Knut Kaasen. *Formalisme i komplekse tilvirkningskontrakter*. I: Festskrift til det Danske Selskab for Byggeret. København, 2009. s. 163-184.

Kaasen (2010) : Knut Kaasen. *Prosjektintegrert megling: et godt løsningsmiddel?* I: På rett grunn – Festskrift til Norsk Forening for Bygge- og Entrepriserett. Jan Einar Barbo og Lasse Simonsen (red.), 2010. s. 286-303.

Langemark (2009) : Jesper Langemark [et al.]. *Iterative prosjektmodeller og kontrakter*. Danmark, 2009.

Larman (2004) : Craig Larman. *Agile and iterative development: a manager's guide*. Boston, Mass., 2004.

Larman (2003) : Craig Larman og Victor R. Basili. *Iterative and incremental developments: A brief history*. I: Computer. Vol. 36 (2003). s. 47-56.

Larman (2010) : Craig Larman og Bas Vodde. *Practices for scaling lean & agile development: large, multisite, and offshore product development with large-scale Scrum*. Upper Saddle River, N.J., 2010.

McConnell (2004) : Steve McConnell. *Code complete*. Redmond, Wash., 2004.

Mestad (1999) : Ola Mestad. *Mangelsreguleringa i norsk fabrikasjonskontrakt 1992 (NF92)*. I: Jussens Venner (1999). s. 272-291. [Sitert 05.05.2014, lovdata.no].

Moløkken-Østvold (2007) : Kjetil Moløkken-Østvold og K. M. Furulund. *The Relationship between Customer Collaboration and Software Project Overruns*. I: Agile Conference (AGILE), 2007, 2007. s. 72-83.

Moløkken-Østvold (2005) : Kjetil Moløkken-Østvold og Magne Jørgensen. *A comparison of software project overruns – flexible versus sequential development models*. I: Software Engineering, IEEE Transactions on. Vol. 31 (2005). s. 754-766.

Moløkken-Østvold (2004) : Kjetil Moløkken-Østvold [et al.]. *A survey on software estimation in the Norwegian industry.* Software Metrics, 2004. Proceedings. 10th International Symposium on, 2004. s. 208-219.

Monsen (2010) : Erik Monsen. *Om reklamasjonsregler, passivitetsprinsipper og realitetsdrøftelser*. I: Jussens Venner nr. 3 (2010). s. 147-203.

Nazarian (2007) : Henriette Nazarian. *Lojalitetsplikt i kontraktsforhold*. Oslo, 2007.

Næss (2012) : Anne Kristine Næss og Espen Frøyland. *Gevinstrealisering av pensjonsreformen i Statens pensjonskasse*. I: Gevinstrealisering og offentlige IKT-investeringer. Leif Skiftenes Flak (red.). Oslo, 2012.

Ottersten (2004) : Ingrid Ottersten og Mijo Balic. *Effektstyrning av IT – Nyttan uppstår i användningen*. Malmö, 2004.

PMI (2013) : PMI. *A guide to the project management body of knowledge (PMBOK guide)*. 5. utg. Atlanta, 2013.

Poppendieck (2003) : Mary Poppendieck og Tom Poppendieck. *Lean software development: an agile toolkit*. Boston, 2003.

Royce (1970) : W. Winston Royce. *Managing the Development of Large Software Systems.* Proceedings of IEEE WESCON, 1970. s. 1-9.

Sandvik (1966) : Tore Sandvik. *Entreprenørrisikoen*. Oslo, 1966.

Schartum (2006) : Dag Wiese Schartum. *Introduction to a Government-based Perspective on Proactive Law*. I: Scandinavian Studies in Law. Vol. 49 (2006). s. 35-52.

Senter for statlig økonomistyring (2010) : Senter for statlig økonomistyring. *Gevinstrealisering – en innføring i planlegging og oppfølging av gevinster* (2010).

Simonsen (1999) : Lasse Simonsen. *Kreditors mangelsbeføyelser – særlig for tilvirkningskontrakten*. I: Jussens Venner nr. 5/6 (1999). s. 305-398.

Sommerville (2011) : Ian Sommerville. *Software engineering*. 9. utg. Boston, Mass., 2011.

Stepanek (2005) : George Stepanek. *Software project secrets: why software projects fail*. Berkeley, 2005.

Sutherland (2013) : Jeff Sutherland og Ken Schwaber. *Scrumguiden* (2013). http://www.scrum.org/scrum-guide. [Sitert 25.03.2014].

Torvund (1997) : Olav Torvund. *Kontraktsregulering – IT kontrakter*. Oslo, 1997.

Turner (1996) : J. Rodney Turner. *International Project Management Association global qualification, certification and accreditation*. I: International Journal of Project Management. Vol. 14 (1996). s. 1-6.

Tvarnø (2002a) : Christina D. Tvarnø. *Loyalitetspligt og partneringaftaler*. I: Julebog 2002. Peter Møgelvang-Hansen (red.). København, 2002a. s. 137-156.

Tvarnø (2002b) : Christina D. Tvarnø. *Partnering – En aftaleretlig udfordring*. I: Retsvidenskabeligt Tidsskrift (Rettid) (2002b). http://law.au.dk/forskning/rettid/artikler/2002/. [Sitert 13.02.2014].

Udsen (2013a) : Henrik Udsen. *IT-ret*. København, 2013a.

Udsen (2013b) : Henrik Udsen. *Ændringsrisikoen i it-udviklingsaftaler – Del 2*. I: Lov&Data nr. 117 (2013b). s. 4-9.

Wateridge (1998) : John Wateridge. *How can IS/IT projects be measured for success?* I: International Journal of Project Management. Vol. 16 (1998). s. 59-63.

Rapporter

Dragsted (2013) : Nicolai Dragsted og Mikael Klint. *It-kontrakter 2012/2013 – Undersøgelse af problemstillinger ved it-kontrakter set fra kunde, leverandør og rådgiver perspektiv* (2013). http://www.bvhd.dk/uploads/tx_mocarticles/It-kontrakter.pdf. [Sitert 20.12.2013].

Entreprenørforeningen – Bygg og Anlegg (2013) : Entreprenørforeningen – Bygg og Anlegg og Hans Chr. Brodtkorb. *Veileder om SAMSPILLSENTREPRISE* (2013). http://www.ebanett.no/getfile.php/Filer/Samspillsentreprise%202013.pdf. [Sitert 13.02.2014].

Meld. St. 23 (2012-2013) : Meld. St. 23. *Digital agenda for Norge – IKT for vekst og verdiskaping* (2012-2013). http://www.regjeringen.no/nb/dep/kmd/dok/regpubl/stmeld/2012-2013/meld-st-23-20122013. [Sitert 29.04.2014].

Rambøll Management Consulting AS (2013a) : Rambøll Management Consulting AS. *IT i praksis 2013 – digitalisering i offentlig sektor* (2013a).

Rambøll Management Consulting AS (2013b) : Rambøll Management Consulting AS. *IT i praksis 2013 – strategi, ledelse, trender og erfaringer i private virksomheter* (2013b).

Standard Norges komité SN/K 534 (2013) : Standard Norges komité SN/K 534. *Nye samarbeidsformer innenfor bygg og anlegg – Er det behov for nye eller reviderte standardkontrakter?* (2013).

Statens pensjonskasse (2012) : Statens pensjonskasse. *Sluttrapport – Investeringsprogrammet Perform 2008-2012* (2012). https://www.spk.no/Global/Perform/spk_sluttrapport_perform_07072012%5B1%5D.pdf. [Sitert 17.12.2013].

VersionOne (2014a) : VersionOne. *8th Annual State of Agile Survey* (2014a). http://stateofagile.versionone.com/. [Sitert 03.02.2014].

Nettsider

Beck (2001a) : *Manifestet for smidig programvareutvikling*. http://agilemanifesto.org/iso/no/ [Sitert 06.02.2014].

Beck (2001b) : *Prinsippene bak Det smidige manifestet*. http://agilemanifesto.org/iso/no/principles.html [Sitert 06.02.2014].

Hoff (2014) : *Er offentlig innovasjon mulig?* http://www.ukeavisenledelse.no/synspunkt/per-morten-hoff/er-offentlig-innovasjon-mulig [Sitert 07.05.2014].

Kommunal- og moderniseringsdepartementet (2011) : *E-valgforsøket 2011*. http://www.regjeringen.no/nb/dep/kmd/prosjekter/e-valg-2011-prosjektet.html [Sitert 25.04.2014].

VersionOne (2014b) : *Benefits of Agile Software Development*. http://www.versionone.com/Agile101/Agile-Software-Development-Benefits [Sitert 03.04.2014].

W3C (2012) : *Web Content Accessibility Guidelines (WCAG) Overview*. http://www.w3.org/WAI/intro/wcag [Sitert 20.05.2014].

Kontrakter

Benefield (2013) : *Flexible Contract*. http://www.flexiblecontracts.com/.

Den Norske Dataforening (2010a) : *PS2000 Smidig*. https://www.dataforeningen.no/ps2000-smidig-smidig-bilagssett.4657604-147780.html.

Den Norske Dataforening (2010b) : *PS2000 Standard*. https://www.dataforeningen.no/it-kontraktsstandarden-ps2000.4598709-134109.html [Sitert 21.01.2014].

Den Norske Dataforening (2013a) : *Dataforeningens kontraktsstandard for oppdragsbasert smidig utvikling av programvare*. http://www.dataforeningen.no/ps2000-sol.5293799-298119.html [Sitert 21.01.2014].

Den Norske Dataforening (2013b) : *PS2000 SOL – veiledning for utarbeidelse og bruk av kontrakten*. http://www.dataforeningen.no/ps2000-sol.5293799-298119.html [Sitert 21.01.2014].

Difi (2013) : *Programutviklingsavtalen (SSA-U)* http://www.anskaffelser.no/verktoy/programutviklingsavtalen-ssa-u.

Difi (2014) : *Smidigavtalen (SSA-S)*. http://www.anskaffelser.no/verktoy/smidigavtalen-ssa-s [Sitert 17.04.2014].

Digitaliseringsstyrelsen (2012) : *K03 Standardkontrakt for længerevarende it-prosjekt baseret på en agil metode*. http://www.digst.dk/Styring/Standardkontrakter/K03-Standardkontrakt-for-agile-projekter [Sitert 13.03.2014].

IKT-Norge (2010) : *Systemutviklingsprosjekt – Standardavtale*. http://ikt-norge.no/butikk/systemutviklingsprosjekt-standardavtale-engelsk-arslisens/.

IT&Telekomföretagen (2012) : *Nytt standardavtal Agila Projekt*. http://www.itotelekomforetagen.se/fakta-och-debatt/nyheter-och-aktuellt/aktuellt-arkiv/nytt-standardavtal-agila-projekt [Sitert 19.01.2014].

Norsk Industri (2007a) : *Norsk Fabrikasjonskontrakt NF 07.*

Norsk Industri (2007b) : *Norsk Totalkontrakt NTK 07.*

Standard Norge (2008) : *NS 8405:2008 Norsk bygge- og anleggskontrakt*.

Standard Norge (2011) : *NS 8407:2011 Alminnelige kontraktsbestemmelser for totalentrepriser.*

Lover

Avtaleloven (1918) : Lov om avslutning av avtaler, om fuldmagt og om ugyldige viljeserklæringer (avtaleloven) av 31. mai 1918 nr. 4.

Bustadoppføringslova (1997) : Lov om avtalar med forbrukar om oppføring av ny bustad m.m. (bustadoppføringslova) av 13. juni 1997 nr. 43.

Håndverkertjenesteloven (1989) : Lov om håndverkertjenester m.m for forbrukere (håndverkertjenesteloven) av 16. juni 1989 nr. 63.

Kjøpsloven (1988) : Lov om kjøp (kjøpsloven) av 13. mai 1988 nr. 27.

Tvisteloven (2005) : Lov om mekling og rettergang i sivile tvister (tvisteloven) av 17. juni 2005 nr. 90.

Forskrifter

Forskrift om offentlige anskaffelser (2006) : Forskrift om offentlige anskaffelser (nr. 402/2006).

Forskrift om universell utforming av IKT-løsninger (2013) : Forskrift om universell utforming av informasjons- og kommunikasjonsteknologiske (IKT)-løsninger (nr. 732/2013).

Personopplysningsforskriften (2000) : Forskrift om behandling av personopplysninger (personopplysningsforskriften) (nr. 1265/2000).

Rettspraksis

Rt. 1917 s. 673
Rt. 1968 s. 783 : Byggmesterdommen.
Rt. 1969 s. 1122 : Block Watne-dommen.
Rt. 1970 s. 1059 : Knutelindommmen.
Rt. 1972 s. 449 : Terrassedommen.
Rt. 1994 s. 626 : Kaiinspektørdommen.
Rt. 1995 s. 1350
Rt. 1998 s. 774
Rt. 2000 s. 199 : Pelsdyrhalldommen.
Rt. 2006 s. 999
RG 1986 s. 89 : (Eidsivating).
LB-2004-8893
LB-2004-11707
LB-2004-102939
TOSLO-2001-11218
TOSLO-2008-126000

Kontraktsregulering av smidig programvareutvikling

Hvilke kontraktsmekanismer kan bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes?

Det er mange it-prosjekter som ikke lykkes. En av årsakene er at særtrekk ved programvareutvikling skaper stor usikkerhet knyttet til programvaren som skal lages. Denne usikkerheten gjør at det ved kontraktsinngåelsen er vanskelig å spesifisere programvarens egenskaper.

Smidig programvareutvikling, en utviklingsmetode som skiller seg radikalt fra tradisjonelle plandrevne utviklingsmetoder, tar disse utfordringene inn over seg. Metoden legger opp til at programvare først skal spesifiseres i detalj underveis i utviklingsprosessen. Målet er å utvikle programvare som er bedre tilpasset kundens behov og dermed gir større nytteverdi.

De tradisjonelle it-kontraktene er ikke tilpasset smidige utviklingsmetoder. Når disse kontraktene likevel brukes i prosjekter som benytter smidig programvareutvikling, øker sjansen for at prosjektet mislykkes. Derfor er det behov for kontrakter tilpasset smidig programvareutvikling.

Et av formålene med denne masteroppgaven, som ble levert ved juridisk fakultet ved Universitetet i Oslo våren 2014, er å bidra i arbeidet med å utforme kontrakter som er egnet for smidig programvareutvikling.

Oppgaven drøfter hvordan alternative måter å spesifisere programvaren på kan bidra til at it-prosjekt som benytter smidig programvareutvikling lykkes. Videre tar den for seg hvordan dette skal reguleres i kontrakten med tanke på risikofordeling og kontraktsbrudd.

Dan Sørensen har hovedfag i informatikk fra Universitetet i Oslo, og har blant annet jobbet med spesifikasjon av programvare, prosjektledelse og kontraktsoppfølging i Kommunal- og moderniseringsdepartementet. Fra desember 2014 jobber han som advokatfullmektig i advokatfirmaet Selmer DA.

9 788272 261565

CompLex