UNIVERSITETET I OSLO

Institutt for informatikk

Praktisk bruk av parprogrammering – et industrielt studie

Masteroppgave

(60 studiepoeng)

Knut Johannes Dahle

26.september 2007

SAMMENDRAG

Parprogrammering er en arbeidsmetodikk som ble gjort kjent gjennom den smidige(*agile*) utviklingsmetodikken *Extreme Programming*(XP). De siste årene har smidige utviklingsmetodikker fått stadig mer oppmerksomhet og utbredelse.

I denne oppgaven er det gjort et case studie på hvordan parprogrammering har blitt brukt i praksis i et stort forretningskritisk prosjekt i Telenor ASA. Systemet som prosjektet omfatter er stort og komplekst, samt at det har blitt utviklet over lengre tid av mange ulike utviklere. Vi ønsket å se om hvordan parprogrammering ble benyttet i praksis i forhold til hvordan arbeidsmetoden blir beskrevet i teorien.

Studiet ble delt opp i en første undersøkelse som samlet inn beskrivende informasjon om subjektene så som utdannelse og erfaring. Deretter er det gjennomført tre undersøkelser i etterkant av tre prosjektleveranser som samlet informasjon om subjektenes bruk av parprogrammering. I tillegg ble det utført observasjoner og intervjuer i prosjektet.

Resultatet viser at parprogrammering ikke passer å bli brukt til alle typer oppgaver. Det virker som det kan være en effektiv måte å la det være opp til utviklerne å bestemme når de ønsker å benytte metoden, men hovedsakelig på komplekse oppgaver og i analyse og designfasen.

Det tyder også på at parprogrammering kan ha en negativ innvirkning på arbeidsmiljøet. Metodikken gjør det blant annet vanskeligere å få individuelt tilpasset arbeidsplassen med pulthøyde, håndleddstøtte, med mer. Det er også resultater som tyder på at parprogrammering kan føre til høyere sykefravær.

FORORD

Først av alt vil jeg takke mine veiledere, Hans Gallis og Erik Arisholm, for deres hjelp og faglige råd.

Jeg vil videre takke familie og venner for deres støtte gjennom mine år som student.

En spesiell takk til Ole Morten Amundsen for hjelp med korrekturlesning.

INNHOLD

FIGURLISTE

LISTE OVER TABELLER

1 INTRODUKSJON

Seksjon 1.1 presenterer bakgrunnen for denne masteroppgaven. I seksjon 1.2 beskriver jeg motivasjonen for oppgaven. Seksjon 1.3 kommer med målsetningen for oppgaven og problemstillingene jeg har valgt. Bidraget til oppgaven blir oppsummert i seksjon 1.4., og aktualiteten til oppgaven blir presentert i seksjon 1.5. Den siste seksjonen gir en oversikt over oppbygningen til resten av oppgaven.

1.1 BAKGRUNN

Parprogrammering går ut på at to utviklere jobber sammen på en datamaskin, med ett sett mus og tastatur. Den ene har rollen som sjåfør og den andre har rollen som navigatør. Sjåføren har kontroll over tastatur og mus. Han har som oppgave å programmere samt forklare navigatøren hva som blir gjort. Navigatøren har som oppgave å gi tilbakemelding, holde oversikten og påpeke feil.

Parprogrammering er ingen ny måte å jobbe på. De grunnleggende konseptene i parprogrammering har vært kjent i mange år[1-4]. Praksisen har fått mye omtale de siste årene spesielt på grunn av introduksjonen av *Extreme programming*[5, 6] (heretter kalt XP) og de mange andre smidige(*agile*) utviklingsmetodene[7, 8].

Antatte fordeler med parprogrammering[9-11]:

- Høyere kodekvalitet og programkvalitet.
- Raskere ferdigstillelse.
- Mindre kostnad.
- Bedre kunnskapsformidling.
- Mer effektiv læring, både i industrien og ved universiteter.

Antatte ulemper med parprogrammering[9-11]:

- Programmer tar flere arbeidstimer å utvikle.
- En i paret kan ende opp med å utføre hele jobben. Dette blir vanskeligere å oppdage for ledelsen.
- Utviklerne blir mindre selvstendige. Stoler mindre på egne løsninger.
- Suksessfull parprogrammering er avhengig av kompatibel personlighet hos utviklerne.
- Utviklerne blir avhengige av hverandre og det er vanskelig å synkronisere felles kalendere.

Smidige metoder og parprogrammering har blitt veldig populært de siste årene. Grunnen til populariteten kan være mange, deriblant et veldig positivt inntrykk av metodene som resultat av flere publikasjoner[5, 6, 10-12]. Mye av den eksisterende forskningen på området er kontrollerte eksperimenter og en stor andel av disse er utført med studenter i universitetsmiljøer. Derfor er nødvendigvis ikke denne forskningen så relevante for industrien .

1.2 MOTIVASJON

Parprogrammering blir ofte omtalt som et enkelt konsept[9]. Lett å bruke og med attraktive fordeler. Et industrielt case studie gir en fin mulighet til å studere om parprogrammering faktisk er så enkelt som mange vil ha det til. Kan det tenkes at parprogrammering ikke bare begrenser seg til at to og to utviklere setter seg sammen på en datamaskin?

Det har blitt utført få studier om hvordan parprogrammering faktisk brukes i praksis, spesielt over tid, og i store og omfattende prosjekter. Mye av den eksisterende forskningen er kontrollerte eksperimenter hvor en stor del er utført med studenter i universitetsmiljøer. Disse studiene er nødvendigvis ikke så relevante for praktisk bruk i industrien. I forhold til kontrollerte eksperimenter kommer det i industrien til en hel del ekstra faktorer som er vanskelig å kontrollere. Samtidig er heller ikke studier av studenters bruk av parprogrammering så relevant. Studiemiljøer avviker ofte en del fra profesjonelle utviklingsorganisasjoner spesielt med tanke på realisme, motivasjon, og erfaring. Dessuten er oppgavene i mange av studiene begrenset i kompleksitet og størrelse.

Det er utført noen case studier av parprogrammering i industrien, men disse har ofte hatt fokus på påståtte effekter ved parprogrammering istedenfor hvordan parprogrammering faktisk brukes og bør bli brukt.

Telenor tok kontakt og ønsket hjelp med innføring og evaluering av deres bruk av parprogrammering i COS prosjektet. Dette er et stort og omfattende prosjekt for et mellomvaresystem i mobildivisjonen. Utviklingen av systemet har pågått siden 1998. I 2005 omfattet systemet mer enn 130 000 kodelinjer og 1 000 000 genererte kodelinjer. Til enhver tid har det jobbet 30 – 60 utviklere i prosjektet. Med andre ord et veldig komplekst system med mange komplekse oppgaver. Høsten 2004 byttet de fra en tradisjonell plandrevet utviklingsmodell til en kombinasjon av *Scrum*[7, 8] og XP. De ønsket en kultur og prosess som utmerket seg innenfor kunnskapsdeling og kommunikasjon med alle involverte parter.

1.3 MÅLSETNING MED OPPGAVEN

Jeg ønsket i denne oppgaven å gjøre et case studie på hvordan parprogrammering har blitt brukt i praksis i et stort forretningskritisk prosjekt i Telenor ASA. Systemet som prosjektet omfatter er stort og komplekst, samt at det har blitt utviklet over lengre tid av mange ulike utviklere.

Gjennom case studiet ønsket jeg å belyse hvordan utviklerne opplever bruken av parprogrammering og hvordan dette utviklet seg over tid. Det blir påstått[13] at blant annet innføringen av ledelsesmetodikken *Scrum* sammen med utviklingsmetodikken XP, hvor parprogrammering er en av praksisene, har hatt stort innvirkning på produktiviteten og kvaliteten i systemet.

Min problemstilling er:

> **Hvordan brukes parprogrammering i praksis i Telenor COS prosjektet i forhold til hvordan bruken beskrives i teorien og eksisterende studier**?

Teorien og den eksisterende forskningen sier en del om hvordan parprogrammering ideeltsett bør benyttes. I og med at teorien og den eksisterende forskningen i stor grad har bakgrunn i kontrollerte eksperimenter og universitetsmiljøer ønsket jeg å se på hvordan parprogrammering faktisk blir brukt i praksis i industrien.

1.4 BIDRAG

Bidraget i denne oppgaven er en beskrivelse av hvordan Telenor COS prosjektet har benyttet parprogrammering. Dette er det eneste studiet som har undersøkt den faktiske bruken av parprogrammering i en såpass tung industriell kontekst. Med bakgrunn i dette har andre organisasjoner som bruker eller vurderer å ta i bruk parprogrammering en ekstra informasjonskilde å basere sine vurderinger og beslutninger på.

I dette studiet kan det tyde på at parprogrammering kun burde brukes på visse typer oppgaver og faser. Eksisterende forskning og resultater fra denne studien viser at parprogrammering fungerer best når det er komplekse oppgaver og i analyse og designfasen.

Det tyder også på at parprogrammering kan ha en negativ innvirkning på arbeidsmiljøet. Metodikken gjør det blant annet vanskeligere å få individuelt tilpasset arbeidsplassen med pulthøyde, håndkeddstøtte, med mer. Det er også resultater som tyder på at parprogrammering kan føre til høyere sykefravær.

1.5 AKTUALITET

Smidige utviklingsmetoder har i de siste årene opplevd økt oppslutning. Dette ser vi blant annet gjennom opprettelse av og økt aktivitet i ulike grupper og fora. Et eksempel her er Oslo XP og Agile Meetup som arrangerer møter og aktiviteter for utviklere, arkitekter og prosjektledere. En ser også at flere og flere IT-selskaper tar i bruk smidige metoder, et eksempel her er Bekk Consulting AS.

1.6 OPPGAVENS STRUKTUR

Første del av oppgaven gir en oversikt over den teoretiske bakgrunnen for denne oppgaven og studiet hos Telenor, før det kommer en gjennomgang av relatert arbeid. Deretter kommer en presentasjon av Telenor og konteksten for studiet. I kapittel 5 ser jeg nærmere på den vitenskapelige tilnærming i oppgaven. Kapittel 6 beskriver resultater fra studiet. Det neste kapitelet inneholder konklusjon og et forslag til videre arbeid etter denne oppgaven. Kapittel 9 inneholder vedlegg, her finner en oversikt over alle spørsmålene som ble brukt i case studiet.

2 TEORETISK BAKGRUNN

2.1 PROSESSFORBEDRING

Prosessforbedring som profesjon fikk sin start gjennom Fredrick W. Taylor sin bok "The Art of Scientific Management" fra 1911[14]. Taylor kikket på hvordan en kunne effektivisere fabrikkindustrien. Arbeiderne skulle være spesialisert for en oppgave og optimalisere hvordan denne oppgaven ble gjort. En av metodene hans var å velge en gruppe med arbeidere og så studere sekvensen av operasjoner som ble utført. Deretter målte han tiden det tok å utføre de ulike operasjonene, for så å eliminere alle feilaktige bevegelser, langsomme bevegelser og unødvendige bevegelser. Til slutt samlet han i en sekvens de raskeste og beste sekvensene. På denne måten innførte han struktur og gjentagelse gjennom observasjon. Teoriene til Taylor fokuserte på produksjonen og lite på arbeidernes velvære og utvikling, og har vært omstridte.

Deming[15] videreutviklet prosessforbedring ved å gjøre den mer vitenskapelig gjennom statistisk prosesskontroll. Deming var en sterk pådriver for bruk av statistikk og målinger i produksjonsprosessene. Prinsippet hans går ut på å utføre målinger underveis og til slutt i prosessen. En prosess er under kontroll dersom variasjonene i måleverdier er innenfor prosessens normale variasjoner. Avvik fra normale variasjoner må analyseres og tiltak for å unngå disse må iverksettes. Men for å lære av erfaringer er det viktig med struktur og at prosessene som blir studert blir gjentatt konsekvent. Deming mente at ved å studere dataene kunne en se om en hadde et forbedringspotensial, og ved å sammenligne data før og etter en endring var innført kunne en se om det var vellykket eller ikke. Dette er bakgrunnen for syklusen planlegg – gjennomfør – sjekk – handle som blir omtalt som *Total Quality Management* (TQM).

Prosessforbedring innen systemutvikling(SPI) er en systematisk metode for å forbedre programvareprodukter[16] gjennom bruk av grunnleggende prinsipper fra TQM. Systemutvikling er derimot en noe uforutsigbar prosess. Derfor skaper dette et ekstra behov for studier av prosjektenes karakteristika, målsetninger, valg av forbedret teknologi, måling og analyser av virkningene. Innen systemutvikling gjøres ofte prosessforbedring(SPI) gjennom forbedring av utviklingsprosessen/modellen til den aktuelle organisasjonen.

Både amerikanske myndigheter og europeiske myndigheter har utnevnt prosessforbedring innen systemutvikling som en av hovedutfordringene i dagens samfunn[17]. Humphrey[18, 19] har vært en av de ledende innen feltet SPI, ved å identifisere de grunnleggende prinsippene for endring og forbedring innen systemutvikling, målorientert måling og gjenbruk av organisasjonens erfaring og kunnskap.

Conradi [17] argumenterer for at målinger er essensielt når en utfører SPI. Dybå[20, 21] prøver å finne faktorer som fører til suksessfulle SPI initiativ. Han mener at organisasjonene må fokusere på å legge til rette en organisasjonskultur som fremmer prosessforbedring.

2.2 SCRUM

Scrum er en tilpassningsdyktig, rask og selvorganiserende utviklingsprosess som stammer fra Japan[7, 8]. Den definerer ikke noen spesifikke utviklingsteknikker for implementasjonsfasene, men fokuserer på hvordan prosjektdeltakerne skal jobbe for å produsere i en konstant endrende kontekst. Det er en empirisk metode som benytter teorier for fleksibilitet, tilpassningsevne og produktivitet.

Hovedidéen i *Scrum* er at systemutvikling involverer flere miljø og tekniske faktorer, som krav, tidsrammer, ressurser og teknologier. Sannsynligheten for at disse faktorene endrer seg er stor. Dette gjør systemutvikling uforutsigbar og komplekst, og krever fleksibilitet av utviklingsprosessen for å håndtere det. Resultatet er bedre kvalitet på leveransene.

Scrum hjelper med å forbedre utviklingspraksisene i organisasjonen, fordi det involvere hyppige aktiviter for å identifisere svakheter og hindringer i utviklingsprosessen samt praksisene som blir benyttet. Schwaber og Beedle[8] foreslår at teamene bør ha 5 – 9 deltakere. Hvis det er flere personer tilgjenglig bør en danne flere team.

2.2.1 PROSESSEN

Scrum prossessen består av tre faser; *pre-game*, utvikling og *post-game*[7, 8].

2.2.1.1 Pre-game

Pre-game fasen består av to delfaser, planlegging og arkitektur/overordnet design. Første del av planlegging er å finne ut av hva det er som skal lages, samt definere prosjektteamene, verktøy, andre ressurser og risikovurdering. Kontroll av potensielle problemer, behov for opplæring, verifisering er også aktiviteter i planleggingsfasen. En liste over arbeid som venter på å bli gjort (*Product Backlog*) blir opprettet (se seksjon 2.2.3.1). Alle kjente krav blir lagt til denne lista. Kravene kan komme fra både kunde, salg, markedsføringsavdeling, kundeservice eller utviklere. Kravene blir estimert og det blir vurdert hvilken prioritet det enkelte krav skal ha. Listen blir kontinuerlig oppdatert med nye og mer detaljerte elementer, samt med nye og mer nøyaktige estimat.

I arkitekturfasen blir det overordnede designet for systemet planlagt på bakgrunn av kravene som ligger i *Product Backlog*. Hvis prosjektet er en utvidelse av et eksisterende system så må en finne ut endringene som trengs å gjøres i systemet for å få inn kravene fra *Product Backlog*. I tillegg må en finne ut hvilke problemer dette kan skape. Et møte blir holdt for å gå igjennom

forslaget til design og implementasjon. Beslutninger blir gjort på grunnlag av resultatet fra dette møtet. I tillegg blir det lagt midlertidige planer for hva de ulike lanseringene skal inneholde.

2.2.1.2 Utvikling

Utviklingsfasen er den smidige delen av *Scrum*. Forvent det uventede. De ulike faktorene som *Scrum* identifiserer og som kan endre seg blir observert og kontrollert gjennom ulike praksiser for hver *Sprint*(se seksjon 2.2.3.3). Dette er faktorer som tid, kvalitet, krav, ressurser, teknologi, verktøy og til og med utviklingsmetode. Istedenfor å se på disse faktorene kun i starten av prosjektet prøver *Scrum* å kontrollere dem hele tiden for å fleksibelt tilpasse seg endringer.

I utviklingsfasen blir systemet utviklet i *Sprint*'er. Dette er gjentagende sykluser hvor ny funksjonalitet blir lagt til systemet. Hver *Sprint* inkluderer de tradisjonelle fasene i systemutvikling; krav, analyse, design, utvikling og leveranse. Arkitekturen og designet til systemet utvikler seg gjennom hver *Sprint*. En *Sprint* kan vare fra to uker til to måneder, samtidig kan det være 2 til 8 *Sprint*'er i et prosjekt før produktet er ferdig for anvendelse. Det kan også være flere team involvert i en leveranse.

2.2.1.3 Post-game

Denne fasen kommer en til når en er enige om at f.eks *Product Backlog* er ferdig. Nå kan ingen flere krav og problemer bli funnet, det kan heller ikke bli opprettet noen. Systemet er nå klart for leveranse og blir forberedt gjennom prosesser som integrasjon, systemtest og dokumentasjon.

2.2.2 ROLLER OG ANSVAR

Det er seks roller i *Scrum* som har ulike oppgaver og formål gjennom prosessen og dens praksiser: *Scrum Master(2.2.2.1)*, Produkteier, *Scrum Team(2.2.2.3)*, Kunde og Ledelsen.

2.2.2.1 Scrum Master

Dette er en ledelsesrolle som er ansvarlig for å sikre at prosjektet blir gjennomført i henhold til praksiser, verdier og regler i *Scrum*. Han er også ansvarlig for å sørge for at progresjonen til prosjektet følger planen. *Scrum Master* arbeider opp mot prosjektteamet, kunden og ledelsen gjennom prosjektet. Han er også ansvarlig for at hindringer og problemer blir fjernet og endret for å sørge for at prosjektteamet jobber så produktivt som mulig.

2.2.2.2 Produkteier

Produkteier er offisielt ansvarlig for å opprette, oppdatere og kontrollere *Product Backlog list*(se seksjon 2.2.3.1). Han blir valgt av *Scrum Master*, Kunden og ledelsen. Han gjør de endelige avgjørelsene om oppgaver i produktets kravliste. Han deltar i estimeringen av *Product Backlog* sine elementer og gjør kravene om til funksjonalitet som skal implementeres.

2.2.2.3 Scrum Team

Dette er prosjektteamet som har myndighet til å gjøre tiltak samt å organisere seg selv slik at de oppnår målene til hver Sprint. *Scrum teamet* er involvert i estimering, opprettelsen av *Sprint Backlog list*(se seksjon 2.2.3.5), gjennomgå *Product Backlog*, og komme med innspill om hindringer som må fjernes fra prosjektet.

2.2.2.4 Kunde
Kunde deltar i oppgaver relatert til produktets kravliste.

2.2.2.5 Ledelsen
Ledelsen har den overordnede beslutningsmyndighet, samtidig som de er ansvarlig for å lage reglene og standardene som skal følges i prosjektet. De deltar også i utarbeidelsen av krav og mål.

2.2.3 PRAKSISER
Scrum har ingen og krever ikke at det skal benyttes noen spesielle utviklingsmetoder/praksiser. Istedenfor krever den ulike ledelsesprosesser og verktøy til å bli brukt i de ulike fasene for å unngå kaos skapt av uforutsette hendelser.

2.2.3.1 Product Backlog
Product Backlog definerer alt som er nødvendig i det endelige produktet basert på nåværende kunnskap. Den definerer jobben som skal gjøres i prosjektet og tilbyr en kontinuerlig oppdatert og prioritert liste over krav for systemet. Elementer i *Product Backlog* kan for eksempel være egenskaper, funksjonalitet, feil, ønskede forbedringer eller teknologioppdateringer. Ulike aktører sånn som kunder, prosjektteam, marked- og salgsavdeling, ledelse og kundeservice, kan opprette elementer i listen. Produkteier er ansvarlig for lista.

2.2.3.2 Estimering
Produkteier og *Scrum* teamene er ansvarlige for estimering. Dette er en gjentakende prosess hvor estimatene blir spesifisert mer nøyaktig ettersom mer informasjon blir tilgjenglig for de ulike kravene.

2.2.3.3 Sprint
Scrum teamet organiserer seg selv til å produsere en ny fungerende delleveranse i en sprint som varer i ca 30 dager. Verktøyene til teamet er *Sprint* planleggingsmøte, *Sprint Backlog* og daglige *Scrum* møter (også kalt standup).

2.2.3.4 Sprint planleggingsmøte
Dette er et tofaset møte som organiseres av *Scrum Master*. Kunden, ledelsen, produkteier, og *Scrum teamet* deltar i første fase av møtet og bestemmer funksjonalitet og mål for neste S*print*. Neste fase av møtet blir holdt mellom *Scrum Master* og *Scrum teamet* som fokuserer på hvordan delleveransen blir implementert igjennom den kommende *Sprint*'en.

2.2.3.5 Sprint backlog

Dette er utgangspunktet for hver *Sprint*. Det er en liste over elementer fra *Product Backlog* som har blitt valgt til implementasjon i den kommende *Sprint*. Elementene blir valgt av *Scrum teamet* sammen med *Scrum Master* og produkteier på *Sprint* planleggingsmøte, basert på prioriteringene og målet til den aktuelle *Sprint*. I motsetning til *Product Backlog* er *Sprint Backlig list* stabil frem til *Sprint*'en er ferdig. Når alle elementene i *Sprint backlog* er ferdige så blir en ny del av systemet levert.

2.2.3.6 Daglig Scrum møte

Daglig *Scrum* møte er for å holde oversikt på progresjonen, samt at de fungerer som planleggingsmøter: hva har blitt gjort siden forrige møte og hva skal gjøres før neste. Problemer og andre ting som teamet støter på blir også diskutert og tatt hånd om i dette møte. Det skal være kort og effektivt. Derfor er det vanlig at alle står oppreist og at det ikke varer lenger enn ca 15min. Det holdes hver dag. Alle hindringer eller problemer blir sett etter, identifisert og fjernet for å forbedre prosessen. *Scrum Master* er ansvarlig for møtene, og i tillegg til *Scrum teamet* kan også ledelsen delta.

2.2.3.7 Sprint gjennomgangsmøte

Siste dag i sprinten presenterer *Scrum Master* og *Scrum* teamet resulatene fra sprinten til ledelsen, kunder, brukere og produkteier i et uformelt møte. Deltakerne vurderer delleveransen og tar beslutning om følgende aktiviteter. Møtet kan resultere i nye krav, og til og med endre retningen for hvordan produktet blir utviklet.

2.3 EKSTREM PROGRAMMERING

Ekstrem programmering (XP) har utviklet seg fra problemene som oppstod rundt tradisjonelle utviklingsmodeller[5]. Det startet som en mulighet til å få gjort jobben gjennom praksiser som hadde vært effektive de siste tiårene[6]. Selv om de ulike praksisene i XP ikke er nye, så har de blitt satt sammen i system i XP slik at de på denne måten danner en ny metodologi for utvikling.

XP sikter til å levere suksessfulle utviklingsprosjekter til tross for uklare eller konstant endrende krav i små og mellomstore team. Korte iterasjoner med små leveranser og rask tilbakemelding, involvering av kunder, kommunikasjon og koordinasjon, kontinuerlig bygging og testing, kollektivt eierskap til koden, begrenset dokumentasjon og parprogrammering er noen av hovedtrekkene til XP.

2.3.1 PROSESSEN

XP består av seks faser: Utforskningsfasen, plannleggingsfasen, iterasjoner før leveranse, ferdigstillingsfasen, vedlikeholdsfasen og dødsfasen.

2.3.1.1 Uforskningsfasen

I utforskingsfasen skriver kunden ned historier som de ønsker inkludert i første leveranse. Hver historie beskriver funksjonalitet som de ønsker i programmet. Samtidig gjør utviklingsteamet seg kjent med verktøy, teknologi og praksiser som skal benyttes i prosjektet. Teknologien og arkitekturen i systemet blir testet gjennom å utvikle en prototype. Denne fasen tar fra et par uker til at par måneder, helt avhengig av hvor kjent programmererne er med teknologien.

2.3.1.2 Planleggingsfasen

I planleggingsfasen prioriteres historiene og det enes om hva som skal komme i første leveranse. Programmererne estimerer hver enkelt historie og en blir så enige om en tidsplan. Tidsplanen for første leveranse overstiger sjeldent to måneder. Planleggingsfasen tar ikke mer enn et par dager.

2.3.1.3 Iterasjoner før leveranse

Denne fasen inkluderer flere iterasjoner før første leveranse. Tidsplanen som en ble enige om i planleggingsfasen blir brutt ned i et antall iterasjoner som hver er på én til fire uker. Første iterasjon lager et system med arkitekturen til hele systemet. Dette gjøres ved å velge historier som gjør at en må bygge hele strukturen. Kunden velger hvilke historier som skal inkluderes for hver iterasjon. Kravtestene lagd av kunden kjøres på slutten av hver iterasjon. Etter siste iterasjon er systemet ferdig.

2.3.1.4 Ferdigstillingsfasen

Denne fasen krever ekstra testing og sjekking av ytelsen av systemet før det kan leveres til kunde. I denne fasen kan en fortsatt finne nye endringer og det må tas en beslutning på om de skal inkluderes i denne leveransen. Endringer som ikke blir gjort i denne leveransen blir dokumentert for senere implementasjon, for eksempel i vedlikeholdsfasen.

2.3.1.5 Vedlikeholdsfasen

Etter at systemet har blitt levert til kunde, må prosjektet både gjøre vedlikehold på systemet som er i drift, samtidig som kan produsere nye iterasjoner. Utvikling av ny funksjonalitet kan redusere farten i denne fasen og det kan være nødvendig å omrokkere teamet, for eksempel ved å få inn nye mennesker med annen kunnskap/fagområder.

2.3.1.6 Dødsfasen

Dødsfasen inntrer når kunden ikke lenger har noen historier som skal implementeres. Det krever også at systemet tilfredsstiller kundens behov ikke bare på funksjonalitet, men også på

andre områder, for eksempel ytelse og pålitelighet. Det er i denne fasen at dokumentasjonen av systemet blir skrevet da det ikke blir gjort noe flere endringer i arkitektur, design eller koden. Dødsfasen kan også nås hvis prosjektet ikke leverer de ønskede resultatene eller at det blir for dyrt med videre utvikling.

2.3.2 ROLLER OG ANSVAR

Det er forskjellige roller i XP for ulike oppgaver og hensikter gjennom prosessen og praksisene.

2.3.2.1 Programmerer

Programmererne skriver tester og holder koden så enkel og eksakt som mulig. En av suksessfaktorene til XP er god kommunikasjon og koordinasjon mellom programmererne og prosjektdeltakerne.

2.3.2.2 Kunde

Kunden skriver historier og funksjonstester. Han bestemmer også når hvert enkelt krav er oppfylt. Kunden prioriterer historiene som skal implementeres, for hver iterasjon.

2.3.2.3 Tester

Testerne hjelper kundene med å skrive funksjonelle tester. De kjører funksjonelle tester regelmessig, formidler resultatene og vedlikeholder testverktøyene.

2.3.2.4 Tracker

Tracker gir tilbakemeldinger i XP. Han sjekker estimatene gitt fra teamet og gir tilbakemelding på hvor nøyaktige de er for å forbedre fremtidige estimater. Han følger også progresjonen i hver iterasjon og evaluerer hvorvidt det er mulig å nå målet innen tidsbegrensingene med de ressursene som er tilgjenglig. Han ser også etter om det trengs noen endringer i prosessen.

2.3.2.5 Coach

Dette er personen som er ansvarlig for hele prosessen. En trygg forståelse av XP er viktig i denne rollen for å kunne guide de andre prosjektdeltakerne i prosessen.

2.3.2.6 Konsulent

En konsulent er en ekstern person som innehar en spesifikk kompetanse som det er behov for i prosjektet. Konsulenten guider teamet til å løse deres spesifikke problem.

2.3.2.7 Sjef (Big Boss)

Sjefen tar avgjørelsene. For å kunne gjøre dette kommuniserer han med prosjektteamet for å holde seg oppdatert på situasjonen, og for å plukke ut eventuelle problemer og feil i prosessen.

2.3.3 PRAKSISER

XP er en samling av ideer og praksiser hentet fra eksisterende metodologier[5].

2.3.3.1 Planning game
Nært samarbeid mellom kunden og programmererne. Programmererne estimerer implementeringen av kundens historier. Kunden bestemmer så hva som skal inngå i leveransen og hvilken prioritering historiene har.

2.3.3.2 Små / korte leveransesykluser
Et enkelt system skal bli laget fort. Minst en gang hver andre til tredje måned. Nye versjoner blir så lansert, fra så ofte som daglig, til minimum en gang i måneden.

2.3.3.3 Metaphors
Systemet defineres av et sett med *Metaphors*, historier, som deles mellom kunden og programmerer. Disse delte historiene guider all utvikling ved å fortelle hvordan systemet fungerer.

2.3.3.4 Enkelt design
Dette går ut på å designe den enklest mulige løsningen som er mulig å implementerer for øyeblikket. Unødvendig kode og kompleksitet fjernes øyeblikkelig.

2.3.3.5 Testing
Systemutviklingen er testdrevet. Enhetstester blir implementert før koden og blir kjørt kontinuerlig. Kunden skriver kravtester.

2.3.3.6 Refactoring
Dette vil si å restrukturere systemet ved å fjerne duplikater, forbedre kommunikasjoner, forenkle og legge til fleksibilitet. Altså å forbedre koden, uten å endre oppførselen.

2.3.3.7 Parprogrammering
To programmerere som jobber sammen på en datamaskin, med ett sett mus og tastatur.

2.3.3.8 Kollektivt eierskap
Alle kan endre en enhver del av koden til enhver tid.

2.3.3.9 Kontinuerlig integrasjon
En kodebit er integrert med resten av kodebasen så fort den er ferdig. Systemet blir integrert og bygget mange ganger om dagen. Alle tester blir kjørt og de må passere for at kodeendringene skal bli akseptert.

2.3.3.10 40 timers uke
Maksimum arbeidsuke på 40 timer. Aldri to uker med overtid på rad. Hvis det skjer blir det behandlet som et problem som må løses. Viktig at alle er opplagte på jobb.

2.3.3.11 Kunde på stedet
Kunden må være tilstede og tilgjenglig for teamet hele tiden.

2.3.3.12 Kodestandarder
Regler for koden som må følges av programmererne.

2.3.3.13 Åpent landskap
Det er foretrukket med et stort rom med små avlukker. Parprogrammerere bør bli plassert sentralt i lokalet.

2.3.3.14 Regler
Teamet har sine egne regler som må følges, men som kan endres til en hver tid. Det må være enighet om eventuelle endringer og følgene må vurderes.

2.4 PARPROGRAMMERING

2.4.1 PROSESSEN
Parprogrammering er en arbeidsmetodikk hvor to utviklere sitter ved siden av hverandre på en datamaskin og kontinuerlig samarbeider på det samme designet, algoritmen, koden eller testen. Den ene i paret blir kalt sjåfør, mens den andre blir kalt navigatør. Sjåføren har kontrollen på tastatur og mus, mens navigatøren følger med på hva sjåføren gjør og holder oversikten[10].

Williams et al.[10] rapporterer om flere positive effekter med parprogrammering. Dette er forbedret programvarekvalitet, kortere utviklingstid, forbedret arbeidsmoral og motivasjon, økt tillit og forbedret samspill, effektiv kunnskapsdeling og forbedret læring. Disse effektene blir forklart med den effekten parprogrammering har på utviklernes adferd. Tilstedeværelsen til partneren utgjør et positivt press til å jobbe hardere, gir mulighet til å lære fra andre, muliggjør tenkning, diskusjon, og enighet om ideer med andre. Noe som igjen fremmer mot til å ta beslutninger, og som øker selvsikkerheten og tilliten til løsningen. I tillegg muliggjør tilstedeværelsen av partneren konstant gjennomgang og kontroll av arbeidet og koden som blir produsert. Dette kan akselerere problemløsningen ved at det er enklere å forklare og diskutere problemer med partneren.

På den andre siden er det også noen påståtte negative effekter med parprogrammering. Siden parprogrammering krever to personer til å jobbe på en oppgave, som ellers kun ville kreve innsatsen til en person, øker den totale innsatsen brukt på oppgaven. Videre blir parprogrammering også kritisert for å ikke ivareta utviklernes privatliv, gjør det vanskelig å konsentrere seg og det kan skape konflikter[22].

2.4.2 ROLLER OG ANSVAR
2.4.2.1 Sjåfør
Hovedoppgaven til sjåføren er å skrive koden, designet eller den testen paret jobber med. Normalt skriver sjåføren på tastaturet, men kan også bruke penn og papir under brainstorming

og diskusjoner rundt design. Sjåføren er mer fokusert på praktiske aspekter ved å gjøre jobben enn å holde oversikt og å se lenger frem enn det som det jobbes med for øyeblikket.

2.4.2.2 Navigatør

Navigatørrollen er mer strategisk. Det trengs både kortsiktig og langsiktig tenkning[10]. Kortsiktige oppgaver som navigatøren gjør er å se på det arbeidet som for øyeblikket blir utført av sjåføren. Dette innebærer å kontinuerlig se etter skrivefeil og syntaks feil med mer. Langsiktige oppgaver som navigatøren utfører er å prøve å se oppgaven det jobbes med i en større kontekst. Og på den måten kunne evaluere om arbeidet tar den rette veien og om sjåføren har en klar visjon om hvordan oppgaven skal løses. Ikke minst må han evaluere om løsningen av oppgaven passer inn i resten av systemet som blir utviklet. På denne måten vil han kunne detektere strategiske feil ved utviklingen. Ofte kan navigatøren gjøre dette bare ved å be sjåføren forklare hva han gjør og hvorfor han gjør det.

I tillegg til å overvåke sjåførens arbeid innebærer navigatør rollen å finne nødvendig informasjon, som relaterte dokumenter og noen ganger å finne alternative måter å løse oppgaven på. Dette for å kunne evaluere om utviklingen kunne vært gjort bedre på en annen måte. Navigatøren bidrar også til å planlegge videre arbeid ved å skrive ned notater og fremtidige oppgaver.

2.4.3 PRAKSISER

2.4.3.1 30 minutter alenetid etter lunsj.

Bruk 30 minutter alenetid etter lunsj for å sjekke mail, ta telefoner og gjøre andre ting som forstyrrer arbeidet i par.

2.4.3.2 Maks 6 timer parprogrammering hver dag.

Parprogrammering er en slitsom måte å jobbe på som krever 100% fokus fra begge parter. Derfor anbefales det å kun jobbe i par 6 timer om dagen slik at partene kan gjøre andre oppgaver som f.eks svare på mail når de ikke sitter sammen.

2.4.3.3 5 minutter pause hver time.

Parprogrammering er en slitsom arbeidsform, og partene blir i blant trøtte. Derfor bør det tas en 5 minutters pause hver time. Slik at de kan hente seg litt kaffe, gå en liten tur, trekke frisk luft, etc.

2.4.3.4 Parrotasjon

Parrotasjon er viktig for at flest mulige utvikler får jobbe litt med hverandre. Slik sørger en for større spredning av kunnskap samt økt læring. Samtidig unngår en at partene i paret begynner å tenke likt. Det er noe som kan skje når et par har jobbet lenge nok sammen. I blant er det par

som fungerer dårlig f.eks pga dårlig kjemi eller stor kunnskapsforskjell, i slike tilfeller kan prosjektet være bedre tjent med at de ikke jobber sammen.

2.4.3.5 Parstjerne

Parstjerne er et enkelt verktøy for å holde rede på hvem som har jobbet i par med hvem og hvem som ikke har jobbet sammen. Det fungerer på den måten at en skriver alle navnene rundt i en ring, deretter trekker en strek mellom navnene på partene i de ulike parene. Etter hvert som en roterer partnere vil det dannes en stjerne av strekene mellom navnene.

2.4.3.6 Alarm

Alarm brukes for å holde rede tiden slik at paret bytter roller jevnlig, for eksempel hvert 20. minutt.

3 Relatert Arbeid

I 1996 ble XP brukt for første gang i C3 prosjektet i Daimler Chrysler[5]. Da ble også parprogrammering presentert som en av nøkkelpraksisene. Siden den gang har mange vist interesse for smidige metoder generelt og parprogrammering spesielt. Til tross for dette er det begrenset med forskning som er gjort på området.

Flere kontrollerte eksperimenter har konkludert med at parprogrammering har mange fordeler fremfor individuell programmering. Dette være seg signifikante forbedringer på forskjellige mål for kvaliteten på programvaren som ble utviklet og redusert utviklingstid. Og dette kun med minimal økning i totalt antall utviklingstimer[11, 23-26]. Det finnes også studier som ikke finner noen positiv effekt for kvalitet eller utviklingstid ved bruk av parprogrammering[27]. Eksperimentene har blitt utført enten med studenter[11, 24, 27] eller med noen få profesjonelle[23, 25].

Resultatene fra de eksisterende empiriske studiene gir verdifull informasjon om parprogrammering og utforsker dets effekt innen forskjellige områder av systemutvikling. Men sammenlikning og analyse mellom studiene er vanskelig, da metrikkene benyttet i de ulike studiene ofte er forskjellige[9, 24, 27]. Et annet problem med metrikkene brukt i noen av studiene er at de ikke samsvarer med prosjekter i industrien, for eksempel om kvalitet er målt ved hjelp av kurskarakterer. Hvordan kan disse bli brukt i en industriell setting?

Arisholm et al.[28] gjorde et eksperiment hvor de så på parprogrammering med tanke på systemkompleksitet og utviklerens erfarenhet. Totalt 295 juniorkonsulenter, konsulenter og seniorkonsulenter fra industrien deltok i eksperimentet. Konsulentene ble leid inn fra konsulentselskaper i Norge, Sverige og Storbritannia i en dag for å delta eksperimentet. I denne studien fant de ut at effekten av parprogrammering er avhengig av en kombinasjon av systemkompleksiteten og utviklerens ekspertise. Eksperimentet viste at juniorkonsulenter mangler kunnskapen til å utføre vedlikeholdsoppgaver på komplekse systemer. Juniorkonsulenter som jobbet i par oppnådde en signifikant økning i nøyaktighet sammenlignet med juniorkonsulenter som arbeidet individuelt. De oppnådde omtrent samme nøyaktighet som en individuell seniorkonsulent.

Herzog[29] beskriver ulike måter å bruke parprogrammering på. Han sier det kan gjøres på mange forskjellige måter, men hovedkategoriene er *blandethet* eller *strenghet*. *Blandethet* er hvorvidt paret jobber sammen over en lengre tidsperiode, *monogam*, eller om det skiftes partner ofte, *blandet*. *Strenghet* er hvorvidt utviklerne må gjøre alt arbeid i par (*streng*) eller om de kan gjøre noe arbeid som for eksempel rutine programmering individuelt (*mild*).

Det finnes selvfølgelig mange grader av *blandethet* og *strenghet*, men hvis graden av *mild* er veldig høy er det diskutabelt hvorvidt det som blir gjort er parprogrammering. Hvis en bruker kun to nivåer, vil en få en matrise som med følgende:

Streng monogam	*Mild monogam*
Streng blandet	*Mild blandet*

Det har ikke blitt identifisert noen fordeler eller ulemper med de forskjellige typene parprogrammering i tidligere forskning. Herzog mener dog at typen parprogrammering en velger vil ha innvirkning på effekten av parprogrammering og hvor sterk denne vil bli. Det kan også tenkes at den ene typen er bedre enn den andre.

4 STUDIEKONTEKST

I dette kapitelet blir det gitt en innføring i bakgrunnen og konteksten for studiet hos Telenor.

4.1 TELENOR

Telenor er Norges største telekommunikasjonsselskap og en av de raskest voksende leverandørene av mobile telekommunikasjonstjenester i verden. Telenors røtter går helt tilbake til 1855. Gjennom 150 år har telekommunikasjon vært vesentlig for utviklingen og byggingen av det moderne norske samfunn. Det fysiske arbeidet har bestått i å bygge infrastruktur og utvikle tjenester. På 1990-tallet ble Telenor omformet fra et statlig monopol til et kommersielt selskap. Det daværende Televerket ble omdannet til et statsaksjeselskap i 1994. I desember 2000 ble Telenor delprivatisert og børsnotert.

Telenors sterke internasjonalisering det siste tiåret er basert på en ledende kompetanse i det norske og nordiske markedet; markeder som teknologisk sett er blant verdens mest avanserte. Norge og Norden har ligget i teten ved utvikling av mobilkommunikasjon og Telenor har vært en pioner innen dette. Manuell mobiltelefonitjeneste ble introdusert i Norge i 1966 – en tjeneste som ble forløperen til det automatiske NMT-systemet som kom i 1981. Dette var da Europas første helautomatiske mobilnett. Den digitale arvtageren, GSM, ble åpnet i 1993, og tredje generasjons mobilnett, UMTS, ble klart kommersielt i 2004.

Telenor hadde ved utgangen av år 2005 eierinteresser i 12 mobiloperatører i Europa og Asia med en total abonnentbase på 82,7 millioner. I 2005 kom 57% av konsernets inntekter fra mobilvirksomhetene. Konsernets inntekter i 2005 var NOK 68,9 milliarder. Ved utgangen av 2005 var 27 600 personer ansatt i Telenor (årsverk) – 16 700 av disse arbeidet utenfor Norge.

4.2 COS KONTEKST

Informasjon om alle Telenor Mobile sine kunder er lagret i alt fra stormaskiner til moderne databasesystemer, spredt over mange ulike hardware og software plattformer.

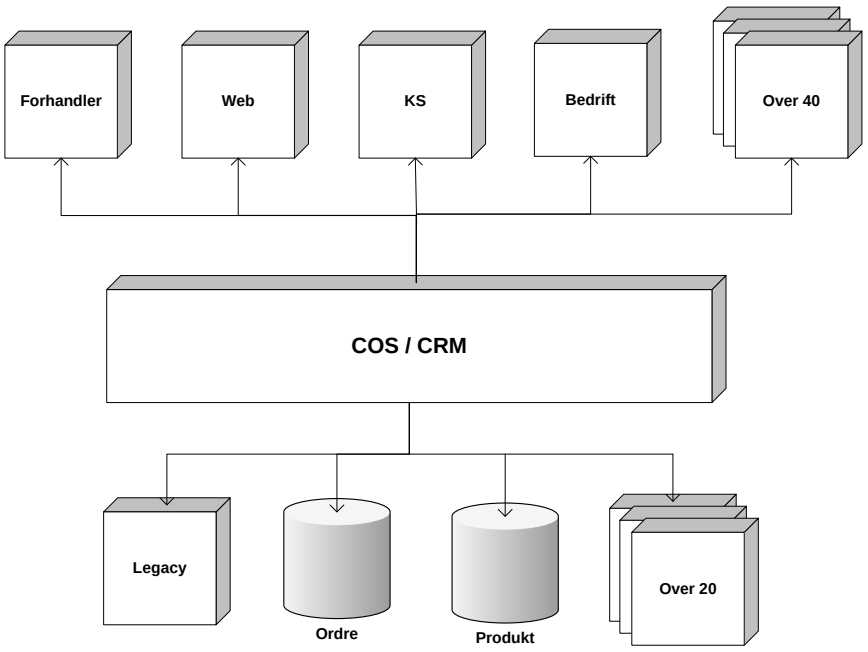
Utviklingen av nye produkter og tjenester er sett på som vesentlig for å opprettholde kundetilfredshet og for å opprettholde Telenor Mobile sin status som en av verdens teknologisk ledende leverandører av mobile telekommunikasjonstjenester. Dette betyr at nye produkter og tjenester kontinuerlig blir utviklet for å skape merverdi for kundene. Det vil si produkter og tjenester utover vanlig talekommunikasjon.

For å dekke behovet til et spekter av ulike klientapplikasjoner som trenger å aksessere data spredt over utallige bakenforliggende systemer, opprettet Telenor Mobile et såkalt mellomvaresystem. Mellomvaresystemet utgjør et veldefinert grensesnitt for

klientapplikasjonene, samtidig som det gjemmer kompleksiteten med å aksessere data fra de bakenforliggende systemene. På denne måten kan nye klientapplikasjoner bli utviklet uten å ha detaljert kunnskap om mellomvaresystemet eller systemene bak. Dette mellomvaresystemet har fått navnet COS, som er en forkortelse for ”Customer Order System”.

4.2.1 COS SYSTEMET

COS systemet[13, 30] er designet for å gi alle klientapplikasjonene et konsistent grensesnitt mot alle de bakenforliggende systemene. Klientapplikasjonene teller mer enn 40 stykk og de leverer tjenester overfor detaljhandel, kundeservice, store bedriftskunder, samt interne funksjoner og annet. De bakenforliggende systemene teller mer enn 30 stykk og omfatter store databaser, stormaskiner, nettverksaktiveringer, datavarehus, osv. Alt sammen logisk sammenkoblet gjennom ulike rutiner og programmer. Systemet er stort og omfattende. Det består av 130K *None Commented Source Statements* (NCSS) og 1000K genererte NCSS. Systemet har mer enn 7 millioner transaksjoner per dag.



FIGUR 1: OVERSIKT OVER COS OG OMLIGGENDE SYSTEMER I TELENOR MOBILE.

Systemet lagrer data om Telenor Mobile sine kunder. Dette inkluderer informasjon om samtaler som danner basis for å fakturere kunden, i tillegg til data vedrørende produkter og tjenester kunden abonnerer på. Dette kan være talepostkasse, tekstmeldinger (SMS), multimediameldinger (MMS), WAP, e-post, fakstjenester, etc. Noe av informasjonen lagres i flere ulike systemer samtidig, som igjen skaper en mulighet for inkonsistens mellom de ulike systemene. Disse dataene trengs å bli brukt på mange ulike måter, fra mange ulike datasystemer og i ulike sammenhenger. For eksempel informasjon om telefonbruken nødvendig for

fakturasystemene for å lage den periodiske telefonregningen. Samtidig trenger kundeservice tilgang til de samme dataene når kunden ringer og klager på regningen. Eller for eksempel når en kunde ønsker å legge til en ny tjeneste på sitt telefonabonnement kan dette gjøres ved å ringe kundeservice eller bestille det direkte gjennom en nettside på internett.

4.2.1.1 Historie

COS prosjektet ble opprettet i 1998, og har omfattet 30 – 60 utviklere til en hver tid. Fra oppstart og frem til i dag har systemet vokst til å bli et stort og omfattende system, bestående av mange delsystemer. Utviklingen skjedde gjennom iterasjoner hvor en utførte kvalitetsikring (QA) mellom hver iterasjon. Altså en typisk plandrevet prosess.

Mesteparten av utviklerne har hatt mer enn 5 års erfaring i tillegg til en universitetsutdannelse. Helt fra starten har det vært innleide konsulenter som har jobbet i prosjektet i tillegg til fastansatte. Sammensetningen av fastansatte og innleide konsulenter har variert, men gjennomsnittlig har det vært en større andel innleide konsulenter som har utviklet på systemet. Det har også vært en høy andel motiverte nyansatte med ønsker om forandringer i systemet.

Etter fem år med vedvarende utvikling av systemet og lavt fokus på vedlikehold ble det mange problemer[13, 30]. Kompleksiteten i systemet økte og andelen mystiske og uventede feil utviklet seg i negativ retning ettersom mengden funksjonalitet ble større. Mye tid ble brukt på feilretting og tester av disse, mens utvikling av ny funksjonalitet var dyrt, risikabelt og sannsynligheten for å introdusere nye feil var stor. Altså mer eller mindre økende entropi. De kom til et punkt hvor noe måtte gjøres og de vurderte det til å være to alternativer. Det ene alternativet var å utvikle alt på nytt fra bunnen av. Det var vanskelig å vite om den investeringen ville lønne seg, og kostnadene ville sannsynligvis være store. Det andre alternativet var å ta utgangspunkt i metodikker for å redesigne det gamle systemet.

Smidige metoder har mange teknikker, slik som automatiske akseptansetester, testdrevet utvikling, kontinuerlig intergrasjon og redesigning, som sørger for at prosjekter ikke ender opp med problemer som beskrevet over. Tanken deres var å benytte seg av disse metodene for å redesigne det eksisterende gamle systemet. En problemstilling de møtte var at dette er metodikker som fungerer for nye prosjekter, men ville de fungere for et eksisterende? Samtidig var det flere utfordringer, blant annet hvordan de skulle få aksept fra ledelsen. Et alternativ kunne være gradvis utfasing av systemet og heller satse på ferdig hyllevare fra andre produsenter. Men hvor skulle de starte? Hva skulle prioriteres? Og hvordan redusere risko?

Dette ble Pareto prosjektet[30]. Hovedmålet med prosjektet var å redusere kostnadene knyttet til vedlikehold og utvikling av ny funksjonalitet. Dette skulle nås gjennom delmålene: 1. Øke produktiviteten med 25 prosent. 2. Redusere feilraten. 3. Målt økning i systemets kvalitet.

Delmål 1 var nok til at prosjektet ble satt i gang. De to andre målene var for kvalitetssikring av endringene som ble gjort. Prosjektet tok utgangspunkt i den italienske økonomen og sosiologen Vilfredo Paretos grunnprinsipp, ofte kalt 80 / 20 regelen[30]. Dette er et grunnprinsipp som har vist seg gjeldene på flere områder. Innen økonomi heter det blant annet at 80 prosent av inntektene dine kommer fra 20 prosent av kundene[31].

Prosjektet hadde tjent seg etter to år, og de kunne allerede etter et halvt år måle effekten blant annet gjennom et egenutviklet verktøy, XRadar[32]. Prosjektets leveranser ble delt inn i flere områder; arkitektonisk kvalitet, kodekvalitet, testkvalitet, kultur og prosess.

Gjennom prosjektet gjorde de mange forbedringer på systemet, blant annet opprydding i sykliske avhengigheter mellom komponenter, fjerning av duplisert kode, detektering og fjerning av ubrukt kode og utvikling av automatiske testrammeverk. For at forbedringene skulle forbli også i nye versjoner av systemet måtte også en endring i kultur og prosess i organisasjonen på plass. De ønsket en kultur og prosess som utmerket seg innenfor kunnskapsdeling og kommunikasjon med alle involverte parter, samt at de ønsket å kvitte seg med de gamle og omfattende rutinene for kvalitetssikring(QA). Dette fordi det tok mye tid og innsats uten at det økte kvaliteten eller minsket feilraten.

De gjennomførte følgende endringer med tanke på kultur og prosess. Endringene ble utført kronologisk.

1. Innførte nytt og moderne system for endringskontroll som støttet smidige praksiser.
2. Inviterte ukentlig alle utviklerne til en halvtimes prat om tekniske og kulturelle spørsmål.
3. Innførte intervjuer av nye konsulenter med fokus på utvalgte emner.
4. Innførte testdrevet utvikling.
5. Byttet ut og fjernet utviklere og arkitekter som fremmet en uønsket kultur.
6. Innførte utviklingsprosessen *Scrum*[7], med teknikker som korte iterasjoner, mer kommunikasjon med kunder og daglige ”stand-up” møter.
7. Innførte parprogrammering som utviklingsprosess for å avløse de gamle rutinene for kvalitetssikring(QA).
8. Startet ukentlige møter hvor utvalgte utviklere fra de ulike prosjektgruppene utvekslet praktisk informasjon.

4.2.1.2 Teknologi

COS systemet er et javabasert system, som kjører på applikasjonsservere fra Weblogic. Systemet omfatter over 8000 klasser fordelt på ca 160 000 linjer med kode, 2-300 database tabeller, 1500

tjenester og 2000 database prosedyrer. Integrasjon gjøres via mange forskjellige teknologier, blant annet meldingskøer, databaser, webtjenester, CORBA, *Screen Scraping Cobol* -systemer, og Batch løsninger.

Utviklingsverktøyet som blir benyttet varierer mellom Eclipse og IntelliJ for javautviklingen. Utviklerne har anledning til å velge det verktøyet som de selv ønsker. Det benyttes også en del andre utviklingsverktøy til de oppgavene som det passer til, f.eks dbArtisan for database, *vi* for Batch programmering og lignende.

Før ble verktøyet Ant benyttet for bygging av prosjektet, men i COS 16 ble dette byttet ut med et nyere verktøy, Maven.

Til måling av kvalitet i systemet benytter de et egenutviklet verktøy kalt Xradar[32]. Xradar bruker flere åpenkildekode verktøy for å analysere kildekoden. Xradar tar resultatene fra disse programmene, sammenfatter det og fremstiller det på en oversiktlig måte i ulike abstraksjonsnivåer. Slik at de ulike interessentene får presentert dataene riktig for seg. Siden prosjektet baserer seg så mye på åpen kildekode som det gjør så har Telenor valgt å legge det ut i åpen kildekode under BSD-lisensen.

4.2.2 COS PROSJEKTET

4.2.2.1 COS teamet

COS har hele tiden bestått av mange prosjektdeltakere, 30 - 60. For å organisere alle disse har prosjektet blitt delt opp i ulike team(EDC, Tjobing! 15, Tjobing! Team I, Tjobing! Team II, Katinekatt, Katana, FakturaKontroll, Teknisk og COSMaven), som i dette studiet(COS15, COS15.1, COS16) varierte fra 2 til 11 medlemmer. De ulike teamene hadde egne oppgaver og fulgte i blant egne iterasjonssykluser. Teamene var som regel midlertidig inndelt og prosjektdeltakerne roterte mellom de ulike teamene. På denne måten fikk alle jobbe med alle, og en fikk variasjon i prosjektet.

4.2.2.2 COS prosessen

COS prosjektet ble delt opp i to *Sprint*'er pr leveranse, unntatt COS 16 hvor det ble delt i tre.

Ved innføringen av parprogrammering innførte de også noen regler for bruk av arbeidsmetodikken:

- Arbeid alene skal ikke overstige 25%.
- 30 minutter alenetid etter lunsj.
- Maks 6 timer parprogrammering hver dag.
- 5 minutter pause hver time.
- Ikke lov å fise mens en parprogrammerer ;)

Dette var regler som hovedsakelig var basert på erfaring fra andre brukere av parprogrammering og de hadde som mål å føre til en effektiv bruk. De benyttet også parstjerne for å holde rede på hvem som har jobbet med hvem, og eventuelt hvem som ikke hadde jobbet sammen.

Når parprogrammering ble innført fikk utviklerne i større grad beskjed om å benytte praksisen. Etter hvert som de brukte det og ble mer kjent med praksisen ble det mer frivillig for hvert enkelt, men de prøvde fortsatt å dele opp slik at to og to utviklere samarbeidet om oppgavene. På denne måten styrte parene mye mer hvorvidt de synes det var hensiktsmessig å sitte sammen og når de fant det bedre å jobbe individuelt.

5 VITENSKAPELIG METODE

Empirisk forskning er forskning basert på verktøy som observasjoner, refleksjon og eksperimentering for å gjøre vitenskapelige framsteg[33]. Empiriske studier spiller en viktig rolle både for å utvikle og å teste teorier i forskningen[34], og er viktig for å ta beslutninger i prosessforbedringer for industrien[35].

Eksperimenter og case studier er de vanligste forskningsmetodene. Hver metode har sine fordeler og ulemper, avhengig av tre betingelser[36]:

- Hvilken type forskningsspørsmål som blir stilt.
- Hvilken grad av kontroll forskeren har over påvirkende faktorer.
- Hvilken grad en fokuserer på nåtid eller historiske hendelser.

5.1 EKSPERIMENTER

Eksperimenter er normalt utført i omgivelser som gir høy grad av kontroll. Eksperimenter er foretrukket når en forsker kan manipulere adferden direkte, presist og systematisk. Subjektene blir gruppert tilfeldig. Målet er å manipulerer en eller flere faktorer og kontrollere alle de andre faktorene[35].

Arisholm[34] sier at for å oppnå full kontroll, er formelle eksperimenter ofte små. Dette utgjør et problem når en prøver å overføre det til prosjekter i virkeligheten.

5.2 CASE STUDIER

Yin[36] definerer et case studie som følger:

1. Et case studie er en empirisk undersøkelse som utforsker et nåværende fenomen i sin virkelige kontekst. Spesielt når grensene mellom fenomen og kontekst ikke er innlysende.
2. Et case studium
    a. har å gjøre med situasjonen hvor det vil være mange flere faktorer av interesse enn det finnes målepunkter.
    b. er avhengig av flere kilder for bevis, hvor dataene må underbygge hverandre triangulært.

c. drar nytte av tidligere utviklede teorier for å finne ut hvordan data innsamling og analyse skal utføres.

Yin[36] sier også at case studie som forskningsmetode er å foretrekke når det er "hvordan" eller "hvorfor" spørsmål, og når de relevante faktorene ikke kan manipuleres eller kontrolleres.

Case studier blir kategorisert i to strategier, kvalitativ og kvantitativ. Case studier som kvalitativ strategi undersøker og evaluerer et program, en aktivitet eller en prosess gjennom en gitt tidsperiode[33, 36]. Case studier som kvantitativ strategi er basert på antagelsen om at en kan identifisere noen målbare egenskaper ved et produkt eller en prosess som en forventer vil endre seg som resultat av metoden/verktøyet en ønsker å evaluere.

Case studiets styrke ligger i dens mulighet til å benytte data og informasjon fra mange ulike kilder, inkludert direkte observasjoner og intervjuer med involverte personer[36]. Dybå[37] sier at styrken med case studie er dets mulighet til å fange virkeligheten mer detaljert og å kunne analysere flere variabler enn det er mulig med andre metoder.

5.3 KVALITATIV VERSUS KVANTITATIV

Kvantitativ strategi bygger på å utvikle metrikker som kan bli brukt til å beskrive fenomenet som er under observasjon. Dataene kan deretter bli analysert ved hjelp av statistiske teknikker. Vitenskapen har i stor grad påvirket bruken av denne strategien i studier av informasjonssystemer[38]. Kvalitativ strategi gjør det mulig for forskeren å beskrive fenomenet på en annen måte. Denne strategien søker å finne andre måter å samle og analysere data på enn kvantitativ strategi[38]. Miles & Huberman[39] sier at kvalitative forskning som regel er basert på ord istedenfor tall. Ord, i form av historier eller beskrivelser, har en konkret og meningsfullt budskap som ofte er mye mer overbevisende for leseren enn sider med summerte tall. Dette fører til at forskeren må bruke mer tid i felten for å samle inn ustrukturerte data og observasjoner i sanntid. Deretter må han bearbeide dataene for å finne meningen i de. Observasjoner kan skje i form av intervjuer, gjengivelse av samtaler og notater fra felten[38]. Resultatene må så bli bearbeidet, strukturert og analyser på en eller annen måte. På den måten er forskerens rolle i kvalitativ undersøkelser noen mer sentral og forstyrrende enn i kvantitative undersøkelser. Den kvalitative forskeren leter etter den individuelle erfaringen og tilordner den sin egen verdi. På den måten må en akseptere et mer subjektivt syn på verden.

5.4 FREMGANGSMÅTE

Studiet ble gjennomført i Norge i tidsrommet november 2004 – august 2005 for prosjektene COS 15 (15.sept 2004 – 13.nov 2004), COS 15.1 (15.nov 2004 – 28.jan 2005) og COS 16 (31.jan 2005 – 20.mai 2005).

Selve studiet ble delt opp i to deler, pre spørreskjema og post spørreskjema. Pre spørreskjema er et spørsmålssett som subjektene besvarte når de ble med i studien. Denne samlet informasjon om subjektet som domenekunnskap, utdanning og erfaring. Post spørreskjema ble besvart av subjektene etter hver leveranse. Den samler informasjon om bruken av parprogrammering for den aktuelle leveransen. Dette er informasjon om hvordan de brukte parprogrammering, hvor mye det ble benyttet, type oppgaver, om reglene satt for prosjektet ble fulgt, og hva som kjennetegnet par som ble oppfattet som gode eller dårlige. Begge settene består av både spørsmål med skalabaserte svar og tekstlige/numeriske besvarelser.

Spørsmålene i pre spørreskjema og post spørreskjema var basert på spørreskjemaene som ble benyttet i parprogrammerings eksperimentet til Arisholm et al[28]. Se seksjon 9.1 for spørsmålene i pre spørreskjema og seksjon 9.2 for spørsmålene i post spørreskjema. Subjektene svarte på spørreskjemaene i gjennom et system kalt SESE[40]. Dette er et elektronisk støtteverktøy for gjennomføring av eksperimenter og case studier innen systemutvikling.

For å få kunnskap om konteksten omkring prosjektet og organisasjonen ble det utført samtaler med prosjekteledere og *Scrum Master.* Deretter ble det gjennomført flere runder med observasjoner i prosjektets lokaler hos Telenor. Det ble først gjennomført en dag med observasjoner for å bli kjent i organsisasjonen og for å finne ut av hva en skulle se etter under senere observasjoner. Basert på informasjon samlet denne dagen ble det utarbeidet et observasjonsskjema for å kunne strukturert samle sammen data og informasjon. Det ble gjennomført tre dager med observasjoner. 1 dag under COS 15.1 og 2 dager under COS 16. Dagene ble fordelt utover i leveransenes tidsrammer slik at en fikk fanget opp de ulike fasene hver leveranse går igjennom.
I løpet av andre leveranse (COS 15.1) ble det holdt en presentasjon hvor midlertidige funn ble presentert for prosjektdeltakerne. Det ble også diskusjoner rundt ulike funn og om kontekst.

5.5 VALIDITET

Case studier kan ikke bli generalisert til en hver situasjon. Formålet med et case studie kan være å bygge bedre modeller for bruk av praksiser i gitte typer organisasjoner. Den aktuelle prediksjonsmodellen er nødvendigvis ikke gyldig utenfor prosjektet eller organisasjonen, men resultatene kan fortsatt være nyttige sett fra en gitt organisasjon. Det faktum at et case studium ikke kan bli generalisert til en hver situasjon trenger nødvendigvis ikke være et problem[34].

For analysen er det tatt utgangspunkt i innledende observasjoner som er gjort under gjennomføringen av studiet. Disse observasjonene er så vurdert og videre analysert mot teori og andre data som er samlet inn. Denne måten å gjøre det på kan påvirke resultatene fordi vurderingene i analysen kan bli subjektive.

6 RESULTATER

6.1 BESKRIVELSE AV SUBJEKTENE I PROSJEKTET

Det er gjort en analyse av hvilke subjekter som deltok i studiet. Antallet subjekter varierte fra de ulike leveransene, noen subjekter kom til mens andre forsvant. Dette har blant annet å gjøre med prosjektsammensetninger for de ulike leveransene, og med hvordan de ulike konsulentene ble disponert. Prosjektet er sammensatt av fastansatte i Telenor og innleide konsulenter fra eksterne selskaper.

Det er noen subjekter som ikke har svart på undersøkelsen. Dette kan skyldes at de ikke har jobbet med parprogrammering eller at de på andre måter har vært forhindret i å svare på spørsmålene. Det er også noen subjekter som har startet, men ikke blitt ferdig med spørsmålene. Det er også et subjekt som har svart på et post spørreskjema, men som ikke har svart på pre spørreskjema. Disse subjektene har blitt valgt bort disse i de videre analysene.

Generelle data	COS 15	COS 15.1	COS 16
Totalt antall subjekter	23	22	31
Antall nye subjekter	23	3	9
Antall subjekter som ikke har svart	3	5	7
Antall subjekter som ikke har jobbet i par	2	0	3
Antall subjekter som har jobbet i par og har svart	18	17	21

TABELL 1: GENERELLE DATA OM ANTALL SUBJEKTER SOM HAR DELTATT.

Her følger en oversikt over fordelingen mellom Telenor ansatte og innleide konsulenter i studien.

Fordeling mellom ansatte og innleide	COS 15	COS 15.1	COS 16
Antall Telenor ansatte	11	11	9
Antall innleide konsulenter	7	6	12

TABELL 2: FORDELING MELLOM ANSATTE OG INNLEIDE

Prosjektet ble for de ulike leveransene delt opp i flere team. Teamene kunne ha en levetid fra et halvt til flere prosjekter.

Prosjektet var delt opp i flere team	COS 15	COS 15.1	COS 16
Antall team	7	3	6

TABELL 3: ANTALL TEAM GJENNOM STUDIET

Telenor COS prosjektet er et prosjekt med erfarne utviklere, og det stilles store krav til nye utviklere som kommer inn i prosjektet.

Programmeringserfaring	COS 15	COS 15.1	COS 16
Gjennomsnittlig antall år med programmeringserfaring	6,11	6,88	7,00
Minimum	3,00	3,00	2,00
1.kvartil	4,25	5,00	5,00
Median	6,00	6,00	6,00
3.kvartil	7,00	10,00	7,00
Maksimum	11,00	12,00	15,00

TABELL 4: SUMMERENDE STATISTIKK OVER PROGRAMMERINGSERFARINGEN TIL SUBJEKTENE PR LEVERANSE.

Utviklerne i COS prosjektet har lang programmeringserfaring, og de har samtidig også stor domeneerfaring.

Domeneerfaring	COS 15	COS 15.1	COS 16
Gjennomsnittlig antall måneder med domeneerfaring	27	25	27
Minimum	8	2	0
1.kvartil	14	11	10
Median	22	21	25
3.kvartil	32	38	36
Maksimum	74	72	77

TABELL 5: SUMMERENDE STATISTIKK OVER DOMENEERFARINGEN TIL SUBJEKTENE PR LEVERANSE.

Ut fra disse dataene ser en at studiet har å gjøre med en organisasjon som primært består av erfarne utviklere, men at det har kommet inn noen nye underveis. Dette stemmer godt overens med andre observasjoner som har blitt gjort.

6.2 RESULTATER FRA SAMTALER OG OBSERVASJONER

Gjennom studiet har det blitt gjort en del observasjoner av hvordan parprogrammering ble brukt og hvordan bruken utviklet seg.

Parprogrammering består av en del praksiser. Det ble observert en del endringer i hvordan disse praksisene ble brukt i forhold til hva en antok ved studiets start.

- Bruken av parprogrammering sank i løpet av studiet. Når de startet opp med parprogrammering i COS 15 var en av retningslinjene at parprogrammering skulle brukes 6 timer hver dag, med 30 minutter alenetid etter lunsj og 5 minutter pause hver time. I starten gav prosjektledelsen klare instrukser om at praksisen skulle følges, men dette ble etter hvert friere. Gjennom COS 15, COS 15.1 og COS 16 utviklet bruken seg til å bli mindre og mindre parprogrammering. Først var det 4 timer pr dag, deretter 2 timer, før det ble at parprogrammering skulle benyttes på de oppgavene der utviklerne følte det var behov for det.

- Parene delte seg opp. Oppgavene ble løst i par – to og to, men i mange situasjoner delte parene seg opp og jobbet hver for seg på sine respektive datamaskiner. De jobbet gjerne sammen i første fase med analyse og design, deretter delte de opp oppgaven og jobbet individuelt.

- Parstjerne ble mindre og mindre brukt. Parene ble mer og mer faste partnere. De ble også sittende mer i faste roller, enten som navigatør eller sjåfør.

Kompleksiteten til oppgavene har påvirkning for når parprogrammering blir brukt.

- Parprogrammering ble benyttet på oppgaver som fra utviklerens synspunkt ble definert som komplekse. Det ble da spesielt brukt under analyse og design. Utviklerne så det som lite effektivt å bruke det på enkle oppgaver og på veldefinerte oppgaver.

Bruk av parprogrammering påvirker arbeidsmiljøet.

- Det ble mange forstyrrelser når de jobbet i par, spesielt av telefoner og møter. Det var heller ikke enkelt å sjekke/svare på e-post.

- Parprogrammering førte til mer støy, spesielt når det er åpent kontorlandskap. En viktig del av parprogrammering er kommunikasjon og derfor må utviklerne snakke sammen.

- Individuell tilpassning av arbeidsdagen med tanke på start og slutt ble vanskeligere. Dessuten var det vanskelig å ta avbrekk for eksempel for å trene.

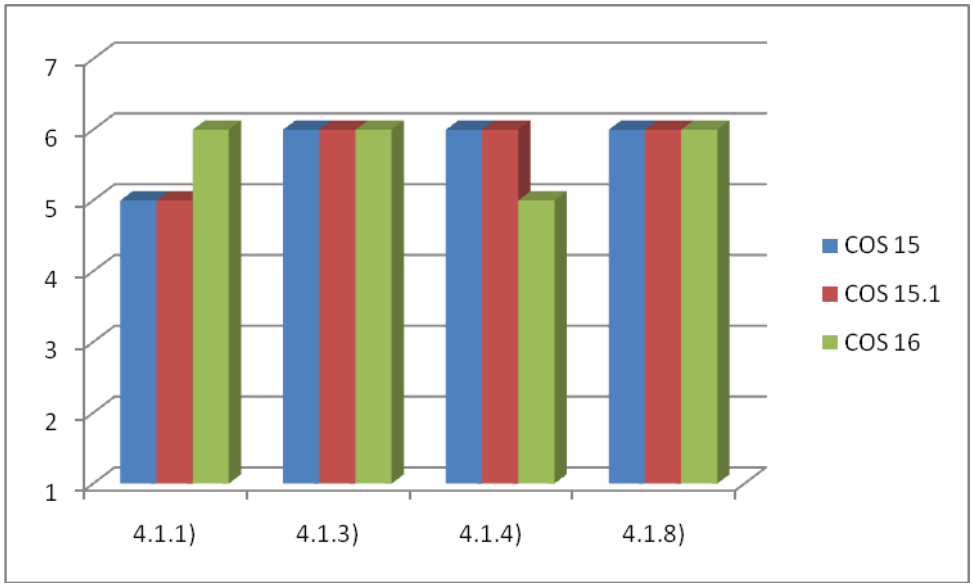
- Individuell tilpassning av arbeidsplassen ble dårligere. Mange som sitter foran datamaskinen hele dagen er avhengig av individuelle løsninger for pulthøyde, håndleddstøtter, etc. Dette var vanskelig når en vekslet på å sitte på sin egen plass og hos partner.

6.3 RESULTATER FRA SPØRREUNDERSØKELSEN

Det har blitt gjort analyser av dataene fra spørreundersøkelsene. Siden undersøkelsene var veldig omfattende er det kun gjort analyser på de spørsmålene som er funnet relevante for problemstillingen i oppgaven.

Spørsmålene som er tatt med i denne analysen har blitt besvart på en skala fra 1 til 7, hvor 1 er *Strongly disagree* og 7 er *Strongly agree*. De originale spørsmålstekstene finner en i seksjon 9.2.

Bruken av parprogrammering kan antas å være avhengig av utviklernes tilfredshet med arbeidsmetoden. Seksjon 4.1) i post spørreskjemaet tar for seg dette området. Figur 2 viser diagrammer av utvalgte spørsmål fra seksjonen.



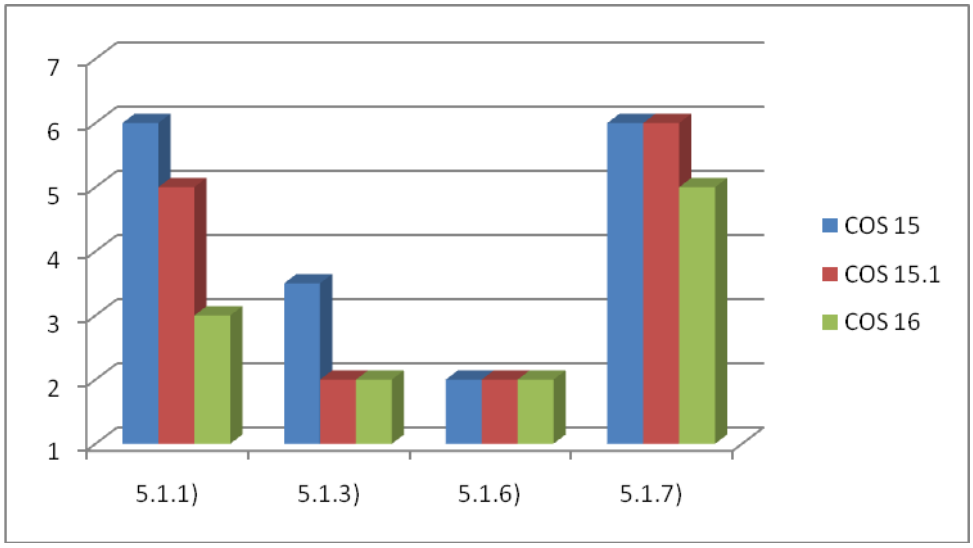
FIGUR 2: GENERELL TILFREDSHET MED PARPROGRAMMERING

Spørsmål 4.1.1) er hvorvidt utvikleren foretrekker parprogrammering fremfor individuell programmering hvis han kunne velge. 4.1.3) er hvorvidt parprogrammering økte utviklerens motivasjon. 4.1.4) er hvorvidt utvikleren lærte nye ting fra de andre utviklerne på teamet når parprogrammering ble benyttet. Til slutt er 4.1.8) om hvorvidt utvikleren liker parprogrammering.

Diagrammet viser at på generelt grunnlag virker tilfredsheten med parprogrammering å være høy. Samtidig er det interresant å se at det er en økning for spørsmål 4.1.1 i COS 16, altså at parprogrammering foretrekkes enda sterkere.

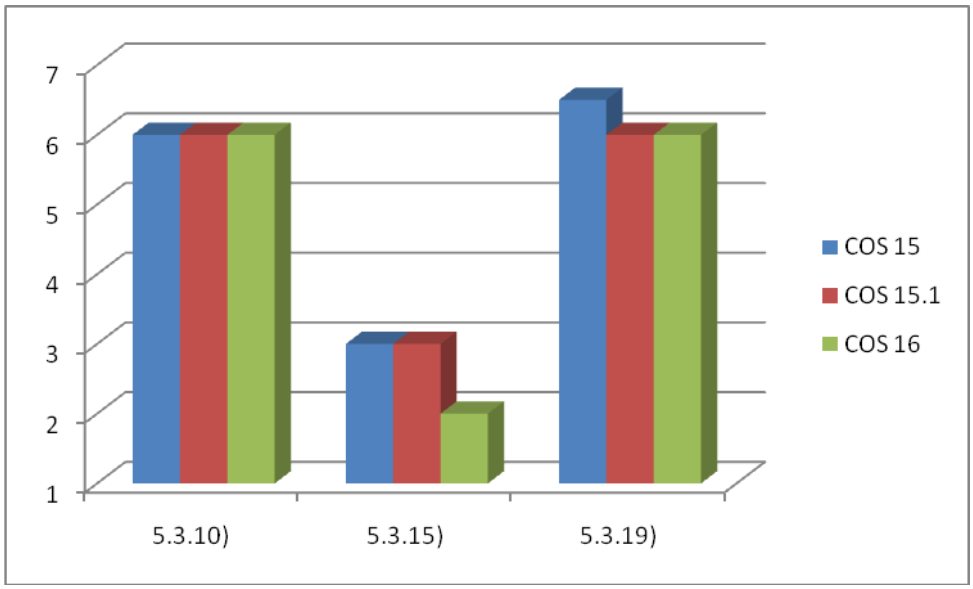
Seksjon 5.1) i post spørreskjemaet tar for seg bruken av kjøreregelene for parprogrammering. Figur 3 viser diagrammer av utvalgte spørsmål for seksjonen.

På spørsmål 5.1.1) om hvorvidt utviklerne har praktisert 30 minutter alenetid etter lunsj ser en nedgang fra COS 15 til COS 16. Samtidig har kjøreregelen om 5 minutter pause hver time blitt lite benyttet(spørsmål 5.1.3). Noe overraskende oppgir subjektene i spørsmål 5.1.6 at de i paret i liten grad har delt oppgavene og utført de individuellt. Dette er i strid med en observasjon omtalt i forrige seksjon. Spørsmål 5.1.7 er om hvorvidt utviklerne i parene regelmessig har byttet mellom å være sjåfør og navigatør. Generelt virker det å ha blitt gjort regelmessig, men med en liten nedgang i COS 16.



FIGUR 3: BRUK AV KJØREGLER FOR PARPROGRAMMERING

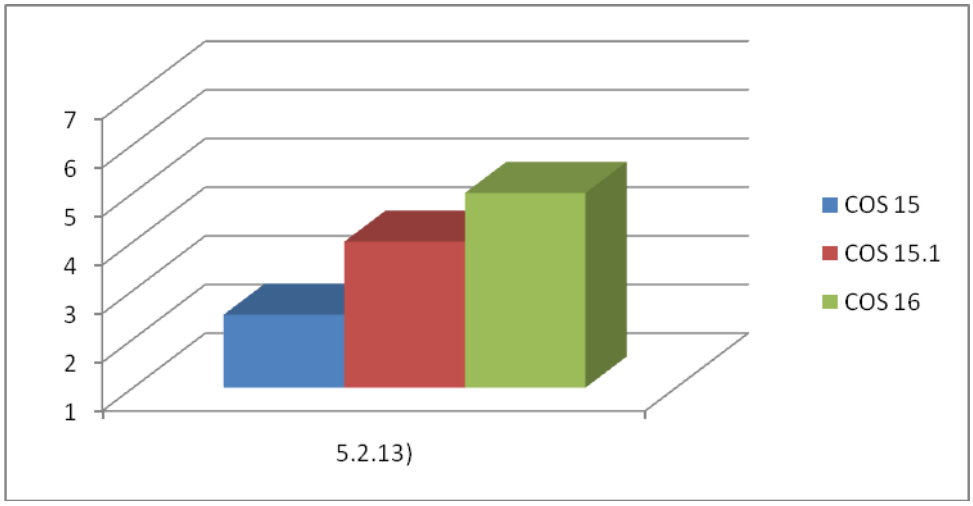
Seksjon 5.3) i post spørreskjemaet tar for seg hvilke oppgaver og situasjoner hvor parprogrammering egner seg brukt. I diagrammet(Figur 4)er det tatt med hvorvidt parprogrammering passer til design(5.3.10), enkle oppgaver(5.3.15) og komplekse oppgaver(5.3.19). Det går ganske tydelig frem her at subjektene mener parprogrammering passer til design og komplekse oppgaver, men ikke så godt til enkle oppgaver. Dette er også i samsvar med observasjonene beskrevet i forrige seksjon.



FIGUR 4: BRUK AV PARPROGRAMMERING TIL ULIKE TYPER OPPGAVER

Det siste diagrammet(Figur 5) viser resultatet for et spørsmål om parprogrammering øker sykefraværet(5.2.13). I COS 15 er denne lav, men den utvikler seg i COS 15.1 og COS 16. Slik at det kan tyde på at subjektene etter å ha brukt parprogrammering en stund mener at det fører til

større sykefravær. Dette er et punkt som nok bør sees i sammenheng med observasjonene om at parprogrammering påvirker arbeidsmiljøet.



FIGUR 5: PARPROGRAMMERING FØRER TIL STØRRE SYKEFRAVÆR

6.4 DISKUSJON

Det er observert at bruken av parprogrammering har sunket fra studiets start til slutt. I studiet viser subjektene preferanser for hvilke type oppgaver de mener parprogrammering egner seg best for. Det er da komplekse oppgaver og i analyse og designfasen. Hvorvidt en oppgave er kompleks eller ikke er avhengig av utviklerens programmeringserfaring og domeneerfaring. Eksperimentet til Arisholm et al.[28] viser at juniorkonsulenter har stor effekt av å bruke parprogrammering på komplekse vedlikeholdsoppgaver, mens det ikke var så stor effekt for seniorer. Subjektene i studien har generelt ganske høy programmeringserfaring og domeneerfaring, men siden prosjektet er så komplekst vil sannsynligvis funnene til Arisholm et al. være gyldige i denne konteksten. I løpet av studiet kom det også inn nye konsulenter i prosjektet. Vi kan anta at disse har hatt stor effekt av å bruke parprogrammering.

Det ble observert at parstjerne ble mindre og mindre brukt. Samtidig som det ble mindre rotasjon av parene. En av grunnene som blir oppgitt for at dette ikke fungerte er utviklerne ble ferdig med sine oppgaver til ulik tid. Oppgavene har ulik lengde, og det var ikke nødvendigvis så mange mulige andre partnere ledige på det aktuelle tidspunktet. Derfor var det ikke så enkelt å bytte.

Det er observert at parprogrammering påvirker arbeidsmiljøet. Mange subjekter følte det ble forstyrrelser pga telefoner og møter. Samtidig som de synes det var vanskelig å sjekke og svare på epost. Williams et al.[10] sier at dette er aktiviteter som kan gjøres utenom kjernetiden for parprogrammering, eller i pauser. Det er i midlertidig vanskelig å styre når innkommende anrop kommer på telefonen. Det er også til en hvis grad ikke så enkelt å legge alle møter til utenom

kjernetid for parprogrammering. Et av subjektene foreslår at ledere burde bli satt i par med hverandre. På den måten kan de unngå å forstyrre andre.

Parprogrammering fører til mer støy. Williams et al.[10] sier at effektive par lager støy fordi de snakker sammen. Stillhet er derimot et signal om et dårlig par. De mener at organisasjonen kan ta grep mot støyen ved hvordan lokalene innredes, for eksempel med egne områder for parprogrammering og egne områder for individuell programmering. I Telenor COS prosjektet er det kun åpenkontorlandskap. Det finnes rom for å jobbe på når en trenger stillhet. Utviklerne i COS prosjektet satt på faste stasjonære datamaskiner rundt i det åpnelandskapet. Det var derfor vanskelig å beskytte seg mot støy.

Individuell tilpasning av arbeidsdagen med tanke på start og slutt ble vanskeligere. Dessuten var det vanskelig å ta avbrekk for eksempel for å trene. Williams et al.[10] sier at løsningen ligger i å balansere mellom ønskene til de ansatte og kravene til prosjektet. Forslaget deres er å operere med en kjernetid da alle utviklerne må være på plass. Uten om denne kjernetiden kan utviklerne gjøre typiske individuelle oppgaver som å sjekke epost, skrive dokumentasjon, lære nye verktøy, undersøke nye teknikker, osv.

Det tyder på at individuell tilpasning av arbeidsplassen var vanskelig. Dette er et område som det faktisk finnes lite forskning på innen parprogrammering. Det er gjort flere studier som ser på hvilken effekt parprogrammering har på folk, inkludert psykologiske og sosiale aspekter. Med tanke på de psykologiske effektene så er det mange studier som rapporterer om økt tilfredshet og glede av arbeidet. For eksempel i [11] rapporterer mer en 90% av subjektene at de likte parprogrammering bedre enn individuell programmering. Den samme tilfredsheten med bruk av parprogrammering ser vi også i dette studiet. Resultatene fra spørreskjemaet viser at subjektene i økende grad mener at parprogrammering fører til høyere sykefravær. Dette kan være på grunn av en kombinasjon av at individuell tilpasning av arbeidsplassen blir vanskelig, samtidig som individuell tilpasning av arbeidsdagen også virker vanskeligere. Samtidig er det studier som viser at parprogrammering er en intens arbeidsform. Det kan tyde på at hvis en utvikler har en litt dårlig dag, er det en del som heller velger å blir hjemme enn å måtte jobbe sammen med en annen person i et par hele dagen.

7 KONKLUSJON OG VIDEREARBEID

I denne oppgaven har vi sett at praktisk bruk av parprogrammering i et stort forretningskritisk prosjekt i Telenor avviker noe fra hva som beskrives i eksisterende teori og studier. Subjektene i studien hadde lang programmeringserfaring og ganske høy domeneerfaring.

Det er klart at parprogrammering ikke er en effektiv arbeidsmetodikk i alle sammenhenger. Parprogrammering virker til å være mest praktisk å benytte for komplekse oppgaver og analyse og designfasen. Det er viktig å merke seg at hvorvidt en oppgave er kompleks eller ikke er avhenging av utviklerens programmeringserfaring og domeneerfaring.

Det er gjort funn som tyder på at parprogrammering påvirker arbeidsmiljøet på en negativ måte, og at den kan føre til høyere sykefravær. Det antas å være mulig å motvirke dette blant annet ved å la utviklerne selv få bestemme om de orker og/eller har behov for å bruke parprogrammering.

Siden det i dette studiet er samlet inn veldig mye data, vil det i et viderearbeid vært naturlig å analysert dataene grundigere enn hva som er gjort i denne oppgaven. I et viderearbeid ville det også vært veldig spennende å sett enda nærmere på parprogrammering sin påvirkning på arbeidsmiljøet.

8 REFERANSER

1. Constantine, L.L., *Constantine on Peopleware*. 1995, New Jersey: Prentice-Hall.

2. Coplien, J.O., *A Generative Development-Process Pattern Language*, in *Pattern Languages of Program Design*, J.O. Coplien and D.C. Schmidt, Editors. 1995, Addison-Wesley: Massachusetts, USA. p. 183-237.

3. Flor, N.V. and E.L. Hutchins. *Analyzing Distributed Cognition in Software Teams: A Case Study of Team Programming During Perfective Software Maintenance*. in *Fourth Workshop on Empirical Studies of Programmers*. 1991. New Brunswick, NJ, USA: Ablex Publishing Corporation.

4. Weinberg, G.M., *The Psychology of Computer Programming*. 1971, New York: Van Nostrand Reinhold Company.

5. Beck, K., *Embrace Change with Extreme Programming.* IEEE Computer, 1999. **32**(10): p. 70-77.

6. Beck, K., *Extreme Programming Explained*. 2000: Addison-Wesley.

7. Abrahamsson, P., et al., *Agile software development methods. Review and analysis*. 2002, VTT Electronics: Espoo.

8. Schwaber, K. and M. Beedle, *Agile software Development With Scrum*. 2002, Upper Saddle River, NJ: Prentice-Hall.

9. Gallis, H., E. Arisholm, and T. Dybå. *An Initial Framework for Research on Pair Programming*. in *2003 ACM-IEEE International Symposium on Empirical Software Engineering (ISESE 2003)*. 2003. Rome, Italy.

10. Williams, L. and R. Kessler, *Pair programming illuminated*. 2002, Boston, Mass.: Addison-Wesley. XXI, 265 s.

11. Williams, L., et al., *Strengthening the Case for Pair Programming.* IEEE Software, 2000. **17**(4): p. 19-25.

12. Williams, L.A. and R.R. Kessler, *All I Really Need to Know About Pair Programming I learned in Kindergarten.* Communications of the ACM, 2000. **43**(5): p. 108-114.

13. Kvam, K., R. Lie, and D. Bakkelund, *Legacy System Exorcism by Pareto's Principle*. 2005.

14. Taylor, F.W., *The Priniciples of Scientific Management.* 1911.

15. Deming, W.E., *Out of the crisis: quality, productivity and competitive position*. 1986, Cambridge: Cambridge University Press. xiii, 507 s.

16. Conradi, R., et al., *Software Process Improvement - Results and Experiences from the Field|*. 2006, Berling Heidelberg: Springer-Verlag.

17. Conradi, R., et al. *Lessons Learned and Recommendations from Two Large Norwegian SPI Programmes*. in *Software ProcessTechnology, Ninth Internationla Workshop, EWSPT'2003*. 2003. Helsinki, Finland.

18. Humphrey, W.S., *Managing the software process*. 1989, Reading, Mass.: Addison-Wesley. XVIII, 494 s.

19. Humphrey, W.S., *Managing technical people: innovation, teamwork, and the software process*. 1997, Reading, Mass.: Addison-Wesley. XIX, 326 s.

20. Dybå, T., *An Instrument for Measuring the Key Factors of Success in Software Process Improvement.* Empirical Software Engineering, 2000. **5**(4): p. 357-390.

21. Dybå, T., *An Empirical Ivestigation of the Key Factors for Success in Software Process Improvement.* IEEE Transactions on Software Engineering, 2005. **31**(5): p. 410-424.

22. Stephens, M., *Extreme Programming Refactored*. 2003.

23. Lui, K.M. and K.C.C. Chan. *When Does a Pair Outperform Two Individuals?* in *Extreme Programming and Agile Processes in Software Engineering*. 2003. Genova, Italy: Springer-Verlag.

24. Müller, M.M., *Two controlled experiments concerning the comparison of pair programming to peer review.* Systems and Software, 2005. **78**(2): p. 166-179.

25. Nosek, J.T., *The Case for Collaborative Programming.* Communications of the ACM, 1998. **41**(3): p. 105-108.

26. Williams, L.A., *The Collaborative Software Process*, in *Department of Computer Science*. 2000, University of Utah: Utah, USA.

27. Nawrocki, J. and A. Wojciechowski. *Experimental Evaluation of Pair Programming*. in *European Software Control and Metrics (Escom)*. 2001. London, England.

28. Arisholm, E., et al., *Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise.* IEEE Transactions on Software Engineering, 2007. **33**(2): p. 65 - 86.

29. Herzog, C.J., *Pair programming and learning*, C.J. Herzog: Oslo. p. 135 s.

30. Kvam, K., R. Lie, and D. Bakkelund. *Cynical Reengineering*. in *XP2004 Procedings*. 2004.

31. Kvam, K., R. Lie, and D. Bakkelund. *A Tool for Cynical Reengineering*. in *rOOts 2004 Procedings*. 2004.

32. *Xradar*, in *Xradar*.

33. Mohagheghi, P., *The impact of software reuse and incremental development on the quality of large systems*. 2004, Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology: [Trondheim]. p. XII, 272 s.

34. Arisholm, E., *Empirical assessment of changeability in object-oriented software*. 2001, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo: [Oslo]. p. VIII, 174 s.

35. Wohlin, C., *Experimentation in software engineering: an introduction*. 2000, Boston: Kluwer. XX, 204 s.

36. Yin, R.K., *Case study research: design and methods*. 2003, Thousand Oaks, Calif.: Sage. XVI, 181 s.

37. Dybå, T., *Enabling software process improvement: an investigation of the importance of organizational issues*. 2001, Department of Computer and Information Science, Faculty of Physics, Informatics and Mathematics, Norwegian University of Science and Technology: Trondheim. p. XVIII, 332 s.

38. Cornford, T. and S. Smithson, *Project research in information systems: a student's guide*. 2006, Basingstoke: Palgrave Macmillan. XII, 201 s.

39. Miles, M.B. and A.M. Huberman, *Qualitative data analysis: an expanded sourcebook*. 1994, Thousand Oaks, Calif.: Sage. XIV, 338 s.

40. Arisholm, E., et al. *SESE - an Experiment Support Environment for Evaluating Software Engineering Technologies*. in *Tenth Nordic Workshop on Programming and Software Development Tools and Techniques (NWPER'2002)*. 2002. Copenhagen, Denmark.

9 VEDLEGG

9.1 SPØRSMÅL PRE QUESTIONNAIRE

3.1) Work experience - number of years
- 3.1.1) *Number of years professional programming experience?
- 3.1.2) *Number of years total work experience?

3.2) Education - Number of credits
- 3.2.1) *How many credits (vekttall) of programming-related (computer science) education do you have?
- 3.2.2) *How many credits (vekttall) of education do you have in total (inlcuding computer science)?

3.3) General Programming Skill
- 3.3.1) *How do you rate your programming skills? (1 = Novice, 5 = Expert)

3.4) Specific Programming Skill - Java
- 3.4.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)
- 3.4.2) *Estimate approximately how many lines of code you have written in this programming language:

3.5) Specific Programming Skill - C++
- 3.5.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)
- 3.5.2) *Estimate approximately how many lines of code you have written in this programming language:

3.6) Specific Programming Skill - C#
- 3.6.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)
- 3.6.2) *Estimate approximately how many lines of code you have written in this programming language:

3.7) Specific Programming Skill - Simula
- 3.7.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)
- 3.7.2) *Estimate approximately how many lines of code you have written in this programming language:

3.8) Specific Programming Skill - SmallTalk

3.8.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)

3.8.2) *Estimate approximately how many lines of code you have written in this programming language:

3.9) Specific Programming Skill - Pascal

3.9.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)

3.9.2) *Estimate approximately how many lines of code you have written in this programming language:

3.10) Specific Programming Skill - Python

3.10.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)

3.10.2) *Estimate approximately how many lines of code you have written in this programming language:

3.11) Specific Programming Skill - C

3.11.1) *How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)

3.11.2) *Estimate approximately how many lines of code you have written in this programming language:

3.12) Specific Programming Skill - Other Language I

3.12.1) Specify programming language

3.12.2) How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)

3.12.3) Estimate approximately how many lines of code you have written in this programming language:

3.13) Specific Programming Skill - Other Language II

3.13.1) Specify programming language

3.13.2) How do you rate your programming skill in this programming language? (1 = Novice, 5 = Expert)

3.13.3) Estimate approximately how many lines of code you have written in this programming language:

3.14) Experience - Design methods and notations

3.14.1) *UML/Rational Rose (1 = Novice, 5 = Expert)

3.14.2) *OMT (1 = Novice, 5 = Expert)

3.14.3) *Responsibility-driven design (1 = Novice, 5 = Expert)

3.14.4) *CRC (1 = Novice, 5 = Expert)

3.14.5) *Role Modelling (1 = Novice, 5 = Expert)

3.14.6)	*Structured Analysis and/or Structured Design	(1 = Novice, 5 = Expert)
3.14.7)	*Data-driven Design (incl. relational databases)	(1 = Novice, 5 = Expert)
3.14.8)	*Use case analysis	(1 = Novice, 5 = Expert)
3.14.9)	*Test-driven development	(1 = Novice, 5 = Expert)
3.14.10)	Other Design Methods/Notations I	
	3.14.10.1) Specify Design Method/Notation	
	3.14.10.2) How do you rate your skill in the specified design method/notation?	(1 = Novice, 5 = Expert)
3.14.11)	Other Design Methods/Notations II	
	3.14.11.1) Specify Design Method/Notation	
	3.14.11.2) How do you rate your skill in the specified design method/notation?	(1 = Novice, 5 = Expert)

3.15) Job function and strengths

3.15.1)	*Currently, what is your primary job function?	Several Alternatives
	Requirements engineering / analysis	
	Architecture and design	
	Coding / programming	
	Testing	
	Quality assurance and process development	
	Maintenance	
	Project- / Line management	
	Integration (deployment)	
	Other	
3.15.2)	Primary job function - Other	
	3.15.2.1) If "Other" regarding job function, specify:	
3.15.3)	*In your opinion, what are your main strengths in software development?	Several Alternatives
	Requirements engineering / analysis	
	Architecture and design	
	Coding / programming	
	Testing	
	Quality assurance and process development	
	Maintenance	
	Project- / Line management	
	Integration (deployment)	
	Other	
3.15.4)	Strengths - other	
	3.15.4.1) If "Other" regarding strengths within software development, specify:	

4.1) Individual orientation

4.1.1)	*I often sought advice and opinions from colleagues	(1 = Strongly disagree, 7 = Strongly agree)
4.1.2)	*I preferred to ask colleagues rather than obtaining information from other sources (e.g., internet sites, electronic processguides) when I was stuck on a task	(1 = Strongly disagree, 7 = Strongly agree)
4.1.3)	*I liked to discuss problems/solutions with colleagues	(1 = Strongly disagree, 7 = Strongly agree)
4.1.4)	*I preferred to let others review my code (code inspection)	(1 = Strongly disagree, 7 = Strongly agree)
4.1.5)	*In general, I preferred to work individually on software development tasks	(1 = Strongly disagree, 7 = Strongly agree)
4.1.6)	*I preferred to Work alone in my own office (where I can work undisturbed) Share an office with one colleague (where I easily can communicate/discuss) Work in an office landscape (where I can communicate and listen to others' conversations/discussions)	Only one alternative

4.2) Experience - Pair programming

4.2.1)	*I had extensive experience with pair programming prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.2)	*I had pair programmed with many different persons prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.3)	*I had a lot of experience with pair programming on requirements engineering/analysis tasks prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.4)	*I had a lot of experience with pair programming on architecture/design tasks prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.5)	*I had a lot of experience with pair programming on coding/programming tasks prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.6)	*I had a lot of experience with pair programming on testing tasks prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.7)	*I had a lot of experience with pair programming on integration (deployment) tasks prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.2.8)	*I had a lot of experience with pair programming on maintenance tasks prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)

4.3) Experience - Team members

4.3.1)	*I knew my team members well (prior to COS <release>)	(1 = Strongly disagree, 7 = Strongly agree)
4.3.2)	*I had worked with one (or several) team members on several previous projects (releases) prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)
4.3.3)	*I had pair programmed with one (or several) team members prior to COS <release>	(1 = Strongly disagree, 7 = Strongly agree)

<release> Referring to the COS release number.

* I starten av spørsmålet betyr at svar på det aktuelle spørsmålet var påkrevd

9.2 SPØRSMÅL POST QUESTIONNAIRE

2.1) Demography

	2.1.1)	*What project(s)/team(s) have you primarily participated in during the COS <release> development period?	Several alternatives
	2.1.2)	If "Other" regarding project/team, specify:	
*)	2.1.3)	*How many percent of your total working time did you spend on COS (incl. COS-EDC)	Only one alternative
	2.1.4)	*What IDE(s)/editors have you used in your work during the COS <release> development period?	Several alternatives
	2.1.5)	If other IDE(s)/editors, please specify	
	2.1.6)	*Type of employment	Only one alternative
	2.1.7)	*How many months of experience with COS do you have?	
	2.1.8)	*How many of the daily stand-up meetings did you attend?	Only one alternative
	2.1.9)	*Did you pair program during COS <release> development and/or system testing?	
	2.1.10)	If you did not pair program during COS <release> development and system testing 2.1.10.1) Describe why you did not pair program:	

3.1) Pair programming - general questions

	3.1.1)	*How much of your DEVELOPMENT/SYSTEM TESTING time did you spend working in pairs?	Only one alternative
	3.1.2)	*How much of your TOTAL working time did you spend working in pairs?	Only one alternative

	3.1.3)	*What do you think were the main advantages (what worked well) regarding pair programming in the COS <release> development period?	
	3.1.4)	*What do you think were the main disadvantages (what did not work well) regarding pair programming in the COS <release> development period?	
	3.1.5)	*What kind of tools did you use to facilitate the discussions and the collaboration during the pair programming in the COS <release> development?	Several alternatives
	3.1.6)	Tools - other 3.1.6.1) If "Other", specify:	
*)	3.1.7)	*The total time I spent on pair programming decreased during the release	(1 = Strongly disagree, 7 = Strongly agree)
*)	3.1.8)	If the total time you spent on pair programming decreased or increased, please explain why you think this happened	

4.1)	Pair programming - general satisfaction		
	4.1.1)	*Given a choice, I would prefer pair programming over individual programming	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.2)	*I think it was fun to pair program	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.3)	*Pair programming increased my motivation	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.4)	*I learned many new things from my team members when we pair programmed	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.5)	*I would have solved the tasks better individually (with higher quality in the code)	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.6)	*I would have solved the tasks with fewer defects in the code if I had worked individually	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.7)	*I would have solved the tasks faster individually	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.8)	*In general, I liked pair programming	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.9)	*Pair programming makes fun work more boring	(1 = Strongly disagree, 7 = Strongly agree)
	4.1.10)	*Pair programming makes boring work more fun	(1 = Strongly disagree, 7 = Strongly agree)

4.2 Pair programming - collaboration
)

4.2.1)	*I enjoyed pair programming with all my partners	(1 = Strongly disagree, 7 = Strongly agree)
4.2.2)	*I think it was easy to pair program with my partners	(1 = Strongly disagree, 7 = Strongly agree)
4.2.3)	*I enjoyed the discussions with my partners	(1 = Strongly disagree, 7 = Strongly agree)
4.2.4)	*My partners contributed actively to solve the tasks	(1 = Strongly disagree, 7 = Strongly agree)
4.2.5)	*I think it was difficult to communicate with my partners	(1 = Strongly disagree, 7 = Strongly agree)
4.2.6)	*I think the collaboration (the pair programming) was useful for solving all the tasks during the COS <release> development period	(1 = Strongly disagree, 7 = Strongly agree)
4.2.7)	*In general, I collaborated well with all my partners	(1 = Strongly disagree, 7 = Strongly agree)
4.2.8)	*I have often felt uncomfortable during pair programming	(1 = Strongly disagree, 7 = Strongly agree)

4.3 Pair programming - information and knowledge sharing
)

4.3.1)	*Pair programming was an efficient technique to increase the information- and knowledge transfer	(1 = Strongly disagree, 7 = Strongly agree)
4.3.2)	*Pair programming increased my understanding (knowledge) of the tasks to be solved	(1 = Strongly disagree, 7 = Strongly agree)
4.3.3)	*My partners came up with ideas and solutions which I did not think of myself	(1 = Strongly disagree, 7 = Strongly agree)
4.3.4)	*My partners discovered defects that I was not aware of	(1 = Strongly disagree, 7 = Strongly agree)
4.3.5)	*My partners gave me useful comments and feedback	(1 = Strongly disagree, 7 = Strongly agree)
4.3.6)	*My partners taught me useful things	(1 = Strongly disagree, 7 = Strongly agree)
4.3.7)	If you learned something new from your partners, what did you learn?	
4.3.8)	*What did you mostly discuss with your partners?	

5.1 Pair programming - driving rules
)

5.1.1)	*I have practiced 30 minutes alone time after lunch	(1 = Strongly disagree, 7 = Strongly agree)
5.1.2)	*I have pair programmed 6 hours at most per day	(1 = Strongly disagree, 7 = Strongly agree)
5.1.3)	*I have taken 5 minutes break every hour	(1 = Strongly disagree, 7 = Strongly agree)
5.1.4)	*I have not farted during pair programming?	(1 = Strongly disagree, 7 = Strongly agree)
5.1.5)	*No more than 25% of my time have been spent on individual work (only work regarding developing and system testing)	(1 = Strongly disagree, 7 = Strongly agree)
5.1.6)	*We have divided our work within the pair into assignments to be performed individually	(1 = Strongly disagree, 7 = Strongly agree)
5.1.7)	*I have regularly switched between driving (sitting with the keyboard) and navigating	(1 = Strongly disagree, 7 = Strongly agree)
5.1.8)	*When tired during pair programming I often take breaks (fresh air, coffee or similar)	(1 = Strongly disagree, 7 = Strongly agree)

5.2) Pair programming - Cost and benefits

5.2.1)	*I think pair programming leads to more defects	(1 = Strongly disagree, 7 = Strongly agree)
5.2.2)	*I think pair programming leads to higher-quality code	(1 = Strongly disagree, 7 = Strongly agree)
5.2.3)	*I think pair programming leads to better tests	(1 = Strongly disagree, 7 = Strongly agree)
5.2.4)	*I think pair programming leads to better design	(1 = Strongly disagree, 7 = Strongly agree)
5.2.5)	*I think pair programming leads to more complex code	(1 = Strongly disagree, 7 = Strongly agree)
5.2.6)	*I think pair programming leads to more efficient system development	(1 = Strongly disagree, 7 = Strongly agree)
5.2.7)	*I think pair programming leads to more fun	(1 = Strongly disagree, 7 = Strongly agree)
5.2.8)	*I think pair programming leads to more stress	(1 = Strongly disagree, 7 = Strongly agree)
5.2.9)	*I think pair programming leads to increased work satisfaction	(1 = Strongly disagree, 7 = Strongly agree)

5.2.10)	*I think pair programing leads to more efficient use of development tools	(1 = Strongly disagree, 7 = Strongly agree)
5.2.11)	*I think pair programming leads to me getting more work done	(1 = Strongly disagree, 7 = Strongly agree)
5.2.12)	*I think pair programming leads to the team getting less work done	(1 = Strongly disagree, 7 = Strongly agree)
5.2.13)	*I think pair programming leads to more sick leave	(1 = Strongly disagree, 7 = Strongly agree)
5.2.14)	*I think pair programming leads to more efficient knowledge transfer	(1 = Strongly disagree, 7 = Strongly agree)

5.3) Pair programming - useful contexts

5.3.1)	*Working in pairs is useful when writing new production code	(1 = Strongly disagree, 7 = Strongly agree)
5.3.2)	*Working in pairs is useful when writing and running system tests	(1 = Strongly disagree, 7 = Strongly agree)
5.3.3)	*Working in pairs is useful when writing unit tests	(1 = Strongly disagree, 7 = Strongly agree)
5.3.4)	Working in pairs is useful when working on well-defined tasks	(1 = Strongly disagree, 7 = Strongly agree)
5.3.5)	*Working in pairs is useful when estimating tasks	(1 = Strongly disagree, 7 = Strongly agree)
5.3.6)	*Working in pairs is useful when refactoring code	(1 = Strongly disagree, 7 = Strongly agree)
5.3.7)	*Working in pairs is useful when making system configuration changes	(1 = Strongly disagree, 7 = Strongly agree)
5.3.8)	*Working in pairs is useful when finding bugs	(1 = Strongly disagree, 7 = Strongly agree)
5.3.9)	Working in pairs is useful when working on tasks that are loosely defined	(1 = Strongly disagree, 7 = Strongly agree)
5.3.10)	*Working in pairs is useful when designing	(1 = Strongly disagree, 7 = Strongly agree)
5.3.11)	*Working in pairs is useful when removing bugs	(1 = Strongly disagree, 7 = Strongly agree)
5.3.12)	*Working in pairs is useful when writing important emails and/or documents	(1 = Strongly disagree, 7 = Strongly agree)

	5.3.13)	*Working in pairs is useful when preparing presentations	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.14)	*Working in pairs is useful when giving presentations	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.15)	*Working in pairs is useful when solving simple tasks	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.16)	*Working in pairs is useful when analysing change requests	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.17)	*Working in pairs is useful when completing source code QA checklists	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.18)	*Working in pairs is useful when talking to people in related projects	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.19)	*Working in pairs is useful when solving complex tasks	(1 = Strongly disagree, 7 = Strongly agree)
	5.3.20)	*Rate the usefulness of pair programming in development vs. system testing	Only one alternative
*)	5.3.21)	*Rate the usefulness of pair programming during the release (COS <release> development and/or system test)	Only one alternative

6)	Pair programming - Roles and pair rotation		
6.1)	*What role did you play when pair programming with your partners?		Only one alternative
6.2)	*My partners and I switched roles frequently		(1 = Strongly disagree, 7 = Strongly agree)
6.3)	*Why did you switch / not switch roles often?		
6.4)	The driver role		
	6.4.1)	*Describe in brief the main advantages you experienced being a driver	
	6.4.2)	*Describe in brief the main disadvantages you experienced being a driver	
	6.4.3)	*I performed very well as a driver	(1 = Strongly disagree, 7 = Strongly agree)
	6.4.4)	*I preferred being the driver	(1 = Strongly disagree, 7 = Strongly agree)
	6.4.5)	*I enjoyed being the driver	(1 = Strongly disagree, 7 = Strongly agree)
	6.4.6)	*I got most out of being the driver (compared to being the navigator)	(1 = Strongly disagree, 7 = Strongly agree)

6.4.7)	*The navigator disturbed me when I was the driver	(1 = Strongly disagree, 7 = Strongly agree)
6.4.8)	*The navigator kept me focused when I was the driver	(1 = Strongly disagree, 7 = Strongly agree)
6.4.9)	*When I was the driver I had problems concentrating because of the navigator	(1 = Strongly disagree, 7 = Strongly agree)

6.5) The navigator role

6.5.1)	*Describe in brief the main advantages you experienced being a navigator	
6.5.2)	*Describe in brief the main disadvantages you experienced being a navigator	
6.5.3)	*I performed very well as a navigator	(1 = Strongly disagree, 7 = Strongly agree)
6.5.4)	*I preferred being the navigator	(1 = Strongly disagree, 7 = Strongly agree)
6.5.5)	*I enjoyed being the navigator	(1 = Strongly disagree, 7 = Strongly agree)
6.5.6)	*I became passive when I was the navigator	(1 = Strongly disagree, 7 = Strongly agree)
6.5.7)	*It was difficult to contribute actively as a navigator	(1 = Strongly disagree, 7 = Strongly agree)

6.6) Pair programming - pair rotation

6.6.1)	*How many members did your team consist of?	
6.6.2)	*Did your team rotate pairs?	
6.6.3)	If yes, how many different persons have you paired up with during the COS <release> development period?	
6.6.4)	*Did your team use a "pair star" for tracking pair rotation?	
6.6.5)	If no regarding "pair star", why did you not use it?	
6.6.6)	If yes regarding "pair star", how did it work?	
6.6.7)	*What kind of pair rotation policy did your team apply?	Only one alternative
6.6.8)	*What kind of pair rotation would you prefer in the next Sprint/project?	Only one alternative
6.6.9)	*Pair rotation is necessary in pair programming	(1 = Strongly disagree, 7 = Strongly agree)

7) Pair programming - pair differences

7.1)	*Working in pairs is more productive than working individually		Only one alternative
7.2)	If you have worked in more than one pair:		
	7.2.1)	All pairs I worked in were about equally successful/rewarding/efficient to work in	(1 = Strongly disagree, 7 = Strongly agree)
	7.2.2)	Most pairs I worked in were about equally successful/rewarding/efficient to work in	(1 = Strongly disagree, 7 = Strongly agree)
	7.2.3)	Some (one or more) pairs were clearly more successful/rewarding/efficient to work in than others	(1 = Strongly disagree, 7 = Strongly agree)
	7.2.4)	Some (one or more) pairs were clearly less successful/rewarding/efficient to work in than the others	(1 = Strongly disagree, 7 = Strongly agree)
7.3)	Think about the pair you have worked in this period that you feel has been MOST successful/rewarding/efficient:		
	7.3.1)	I enjoyed pair programming with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.2)	I think it was easy to pair program with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.3)	I enjoyed the discussions with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.4)	My partner in this pair contributed actively to solve the tasks	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.5)	It was difficult to communicate with my partner in this pair	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.6)	The tasks to be solved when I paired with this person was more suitable for pair programming	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.7)	My partner in this pair and I collaborated well	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.8)	I was mostly the driver when pairing with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.9)	I learned a lot from this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.10)	The person I paired with was more experienced than me	(1 = Strongly disagree, 7 = Strongly agree)
	7.3.11)	The person I paired with had more domain knowledge than me	(1 = Strongly disagree, 7 = Strongly agree)

	7.3.12)	My partner in this pair and I had equal programming skills	(1 = Strongly disagree, 7 = Strongly agree)
**)	7.3.13)	*My partner and I split up and solved parts of the task individually	
**)	7.3.14)	If you did split up, describe why you did, and on what kind of tasks	
	7.3.15)	Why do you feel this pair was MOST successful/rewarding/efficient?	
7.4)	Think about the pair you have worked in during this period that you feel has been LEAST successful/rewarding/efficient:		
	7.4.1)	I enjoyed pair programming with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.2)	I think it was easy to pair program with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.3)	I enjoyed the discussions with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.4)	My partner in this pair contributed actively to solve the tasks	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.5)	It was difficult to communicate with my partner in this pair	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.6)	The tasks to be solved when I paired with this person was more suitable for pair programming	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.7)	My partner in this pair and I collaborated well	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.8)	I was mostly the driver when pairing with this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.9)	I learned a lot from this person	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.10)	The person I paired with was more experienced than me	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.11)	The person I paired with had more domain knowledge than me	(1 = Strongly disagree, 7 = Strongly agree)
	7.4.12)	My partner in this pair and I had equal programming skills	(1 = Strongly disagree, 7 = Strongly agree)
**)	7.4.13)	*My partner and I split up and solved parts of the task individually	
**)	7.4.14)	If you did split up, describe why you did, and on what kind of tasks	

7.4.15) Why do you feel this pair was LEAST successful/rewarding/efficient?

8) Pair programming and individual development culture

8.1) Perceived compatibility between pair programming and personal development culture

8.1.1)	*Pair programming is compatible with the way I prefer to develop software	(1 = Strongly disagree, 7 = Strongly agree)
8.1.2)	*Pair programming fits well with how I prefer to work	(1 = Strongly disagree, 7 = Strongly agree)
8.1.3)	*Pair programming is compatible with the way I organize and structure my work	(1 = Strongly disagree, 7 = Strongly agree)
8.1.4)	*Pair programming is compatible with all aspects of my work	(1 = Strongly disagree, 7 = Strongly agree)

8.2) Pair programming - future intentions

8.2.1)	*I will recommend pair programming to my colleagues	(1 = Strongly disagree, 7 = Strongly agree)
8.2.2)	*Given a choice, I would use pair programming more in my daily work	(1 = Strongly disagree, 7 = Strongly agree)
8.2.3)	*In my opinion, the advantages of using pair programming outweigh the disadvantages	(1 = Strongly disagree, 7 = Strongly agree)
8.2.4)	*Given a choice, I would prefer pair programming over individual programming	(1 = Strongly disagree, 7 = Strongly agree)
8.2.5)	*I hope to pair program more in the future (than I have done so far)	(1 = Strongly disagree, 7 = Strongly agree)
8.2.6)	*Pair programming in the future will decrease my motivation	(1 = Strongly disagree, 7 = Strongly agree)

9) Comments and suggestions

9.1) Comments and suggestions - pair programming

9.1.1) When would you prefer to use pair programming?

9.1.2) What do you believe are the main criteria for working as a useful and efficient pair?

9.1.3) Do you have any suggestions on how to improve the pair programming activity (how to make it more efficient/useful)?

9.2) Comments and suggestions - this questionnaire

9.2.1) Do you have any comments or feedback regarding this questionnaire?

If you have any suggestions for improving this questionnaire please write them here (e.g., new questions, questions that were difficult to understand)

*) New question from post questionnaire 15.1
**) New question from post questionnaire 16
<release> Referring to the COS release number.
* I starten av spørsmålet betyr at svar på det aktuelle spørsmålet var påkrevd