PROSJEKTOPPGAVE

XIS810 Spesialemne i systemarbeid

Høgskolen i Hedmark, mai 2004

Hvilke fordeler kan man oppnå ved å bruke parprogrammering framfor individuell programmering, og hva er viktige forutsetninger for å oppnå disse fordelene?

Levert av N.N.

INNHOLDSFORTEGNELSE

1 Innledning

Bedrifter som driver systemutvikling har et stadig behov for å forbedre sine prosesser for å opprettholde og styrke sin posisjon i et krevende marked. Programmering utgjør en svært vesentlig del av systemutviklingen, og det vil derfor i mange tilfeller finnes et relativt stort potensial for forbedring av denne prosessen.

1.1 Valg av oppgave og problemstilling

I denne oppgaven undersøker jeg hvilke fordeler man eventuelt kan oppnå ved å benytte parprogrammering framfor individuell programmering. Fordeler vil i denne sammenheng være positive effekter i forhold til kostnader, tidsforbruk og kvalitet knyttet til utvikling av softwareprodukter. Jeg forsøker også å definere viktige forutsetninger, og hva man bør eller må ta hensyn til for å optimalisere effekten av parprogrammering. Problemstillingen er:

> *Hvilke fordeler kan man oppnå ved å bruke parprogrammering framfor individuell programmering, og hva er viktige forutsetninger for å oppnå disse fordelene?*

Kilder for oppgaven er rapporter fra studier og empirisk forskning, fagartikler og -litteratur, samt erfaringer fra enkeltpersoner og bedrifter.

Da jeg satte i gang med informasjonssøk til oppgaven var jeg opptatt av å finne *kritiske suksessfaktorer* forbundet med PP. Min problemstilling var også utformet ut fra en slik vinkling til å begynne med, men jeg valgte å endre den. Bakgrunnen for dette ligger først og fremst i den dialogen jeg hadde tidlig i skriveprosessen med programmerere og andre med erfaring. Jeg innså, ut fra svarene jeg fikk og også artikler jeg leste, at begrepet *kritiske suksessfaktorer* er for komplekst og for vanskelig å definere til at man uten videre kan trekke konklusjoner om det.

Jeg fant etter hvert ut at *viktige forutsetninger* vil være en riktigere betegnelse, fordi dette er et begrep som gir mer rom for gradering og vurdering ut fra hver enkelt situasjon. Et moment som er svært viktig i en kontekst, kan være mindre viktig i en annen kontekst.

I deler av oppgaven vil man likevel støte på begrepet *kritiske suksessfaktorer*. Dette gjelder fortrinnsvis delkapitler og vedlegg som helt eller delvis bygger på direkte kontakt med enkeltpersoner og bedrifter. Dette er gjort for at fremstillingen skal være korrekt i forhold til spørsmålene som faktisk ble stilt.

Oppgaven avgrenses til å undersøke PP som frittstående teknikk, ikke som del av en spesiell utviklingsmetode.

1.2 Parprogrammering

Parprogrammering er en teknikk som blir stadig mer kjent og utbredt som et alternativ til tradisjonell, individuell programmering. Systemutviklingsmetoden Extreme Programming - XP - er kanskje den nærmeste årsaken til at parprogrammering har fått økt oppmerksomhet i de senere år. I XP inngår parprogrammering som en av tolv disipliner, og går ut på at all kode utvikles av to programmerere som sitter sammen foran en skjerm [1]. Gjennom XP har teknikken blitt definert og fått et navn, men parprogrammering i ulike former har blitt utøvd gjennom mange år uten direkte knytning til en spesifikk metode [2].

1.3 Definisjoner

Parprogrammering er en form for programmering hvor to utviklere sitter sammen foran en datamaskin og samarbeider kontinuerlig om analyse, design, algoritmer, kode og testing. Betegnelsen *driver* og *navigatør* brukes om rollene i paret. Driveren sitter med tastaturet og skriver ned design eller kode, mens navigatøren aktivt observerer arbeidet. Navigatøren ser etter taktiske og strategiske feil, tenker helhetlig og fremover og korrigerer driverens retning i forhold til designet [3].

Programmerer og **utvikler** er brukt som likeverdig benevnelse i oppgaven.

Begrepsliste

Artikkel	Samlebegrep for alle typer skriftlige framstillinger.
Driver	Driver er den i paret som skriver koden.
Junior	Utvikler som har liten eller ingen erfaring som profesjonell.
Lines of Code (LOC)	Begrep i systemutvikling som spesielt er forbundet med produktivitetsmåling
Medgått tid	Klokketid fra oppstart til ferdigstillelse av oppgave, uavhengig av antall programmerere. I de fleste tilfeller synonymt med utviklingstid. Jfr. Produksjonstid.
Navigatør	Navigatøren er den i paret som kontrollerer driverens arbeid, har den helhetlige oversikten og passer på at fokus er på oppgaven som skal løses.
Paring	En direkte oversettelse av begrepet pairing. Det å sette sammen i par. I noen tilfeller også brukt om det å utføre PP.
Produksjonstid/ -timer	Total tid som er brukt til programmering. Hvis et par programmerer i en time, er produksjonstiden to timer. Jfr Medgått tid
Senior	Utvikler som har relativt lang erfaring som profesjonell.
Vedlikehold	Brukes for det engelske begrepet maintenance. I denne sammenhengen vil det si systemforbedringer, videreutvikling og drift av programvare.
Utviklingsbedrift	Bedrift eller avdeling i bedrift som driver med systemutvikling.

Forkortelser som er brukt for å øke lesevennligheten:

CMM	Capability Maturity Modell. (Prosessforbedringsmodell.)
PP	parprogrammering
IP	individuell programmering
LOC	Lines of Code (jfr begrepsliste)
SW	Software

2 Metode og disposisjon

2.1 Metode

Metode for denne undersøkelsen er basert på forelesningsnotater, innspill, foredrag og drøftinger på de to seminarene for kurset.

Informasjonssøk har vært foretatt mot databaser som er tilgjengelige gjennom Høgskolens websider (Biblioteket). Disse databasene inneholdt lite stoff knyttet til min problemstilling, så ytterligere søk på internett var nødvendig. Ulike søkemotorer og –tjenester har blitt benyttet, og aller mest Google. En del databaser krever både registrering og betaling for å få tilgang til artikler og rapporter, men gjennom søk på tittel i Google fikk jeg tak i artiklene på andre nettsteder.

Biblioteket ved Høgskolens avdeling på Rena har vært behjelpelig med å skaffe bøker.

Jeg har i arbeidet med denne oppgaven også innhentet informasjon direkte fra bedrifter, og fra personer som har erfaringer fra PP. Ved søk på internett fant jeg fram til firmaer som bruker eller har brukt teknikken, og sendte en forespørsel til disse (vedlegg B). Tre personer sa seg villig til å komme med innspill ut fra personlige erfaringer (jfr. Punkt 3.9). I tillegg fikk jeg tilsendt en erfaringsrapport fra et firma (jfr. punkt 3.8).

2.2 Disposisjon

Oppgaven er delt i to hoveddeler etter de innledende kapitler. I første del (kapittel 3) presenteres og oppsummeres de mest sentrale kildene jeg har brukt, og relevans og gyldighet vurderes. I andre del (kapittel 4) settes konklusjonene i kildene opp mot oppgavens problemstilling. Andre kilder av relevans trekkes også delvis inn her. Kapittel 5 presenterer noen analogier til PP, og etterfølges av oppgavens konklusjon.

3 Empiri og forskning

I denne delen vil jeg ta for meg kilder som har relevans i forhold til min problemstilling.. Kildene som er vurdert benevnes artikkel og er tildelt et nummer for å forenkle senere referering. Dette gjelder også sammendraget av spørreundersøkelsen i punkt 3.9.

3.1 Artikkel 1 Strengthening the Case for Pair Programming

Forfattere: Laurie Williams, Robert R. Kessler, Ward Cunningham og Ron Jeffries.
Willams og Kessler underviser på universitetsnivå i tillegg til å drive forskning innenfor sitt fagområde. Cunnigham og Jeffries driver konsulenttjenester innenfor systemutvikling. Jeffries tilbyr også kurs i Extreme programming.

Innhold
Forfatterne har selv gjennomført en studie med studenter ved universitetet i Utah og de viser også til foreløpige resultater fra en online undersøkelse blant profesjonelle programmerere. Studenteksperimentet sammenlignet PP med IP. Artikkelen presenterer resultatene og sammenligner også med funn fra en tidligere studie av Nosek [5].
Artikkelen er svært ofte referert i senere forskning og den står åpenbart sentralt innenfor fagområdet PP.

Påstand: PP gir bedre produkter på kortere tid – og gladere og mer fornøyde programmerere.

Hovedmomenter

Bedre produkt på kortere tid

PP særpreges ved at to personer arbeider sammen i hele utviklingsprosessen, noe som gjør at man både får et bedre design på kortere (medgått) tid og færre feil. Resultater fra studenteksperimentet bekrefter dette; parene leverte sine løsninger vesentlig raskere og med færre feil enn de individuelle.
Antall programmeringstimer var imidlertid høyere med PP, men det hevdes at det er flere potensielle sideeffekter ved PP som man kan dra nytte av i industrien:

-feil avdekkes og forhindres tidlig (reduserer kostnader til feilsøk og –retting)
-godt design (enklere vedlikehold)
-læring og kunnskapsdeling på tvers av teamet (krever færre ressurser til kurs etc)

Man vil altså kunne spare inn de økte timekostnadene i et mer langsiktig perspektiv. Dessuten viste undersøkelsen at parene brukte kortere tid for hver oppgave, noe som forklares med at paret trenger noe tilvenningstid.

Mer fornøyde programmerere

I nettundersøkelsen og studenteksperimentet sa henholdsvis 96 og 90 % at de trives bedre med PP enn med IP. I tillegg var de aller fleste tryggere på løsningene sine. Noe av årsaken til økt trivsel er at man får bruke mer tid på å løse utfordrende oppgaver, enn på frustrerende feilsøking. Lagånden som oppstår gjennom PP nevnes også som en positiv faktor.

Viktige forutsetninger

I online-undersøkelsen rapporterte flere at de i visse tilfeller bruker IP i tillegg til PP under implementasjonen, for eksempel for rutinejobber og enklere oppgaver. Det poengteres at i slike tilfeller er det svært viktig at man har brukt PP på analyse og design.
De fleste av deltakerne i undersøkelsen og eksperimentet hadde selv valgt å bruke PP, men det er ikke alle som passer til PP. Det som oftest kan skape problemer er personer med for mye – eller for lite ego. Man skal ikke være for dominerende, men heller ikke for passiv.

Konklusjon

PP gir et produkt med høyere kvalitet og kortere utviklingstid enn IP. Etter en tilvenningstid vil produksjonstiden ved PP kun være ca 15% høyere enn ved IP. I dagens marked vil rask levering, høy kvalitet, reduserte vedlikeholdskostnader og en fornøyd kunde være et sterkt konkurransefortrinn. Disse fordelene oppveier enhver økning i kostnader til produksjonstimer. Ytterligere forskning er nødvendig for å undersøke om programmerere kan bli mer fornøyd også når de i utgangspunktet er motvillige til å bruke PP.

Oppsummering

Artikkelen er skrevet for flere år siden (2000), men mye av essensen i PP dreier seg om forhold som ikke endrer seg hyppig, blant annet kommunikasjon mellom mennesker [6] og problemløsning[9]. Poenget i konklusjonen i forhold til markedet er høyst relevant og derfor anser jeg denne artikkelen som fortsatt aktuell.

Noe skepsis er knyttet til at undersøkelsen er utført med studenter. En gruppe av studenter speiler ikke den kompleksiteten som man vil finne i en gruppe av profesjonelle. Studentene

vil være mer homogene i forhold til faktorer som erfaring, faglig kompetanse, alder etc. I tillegg er de alle i en opplæringssituasjon. Dette kan ha påvirket resultatene i en viss grad i forhold til at risikoen for konflikter i paret reduseres.

Undersøkelsen er basert på å løse fire oppgaver over seks uker, og gir dermed ingen holdepunkter for at effekten vil holde seg over tid og for ulike typer oppgaver. At studien er gjennomført av forfatterne selv kan også ha påvirket resultatene. En del momenter vil likevel være overførbare til næringslivet, noe som også bekreftes av online undersøkelsen.

3.2 Artikkel 2 Pair Programming: Pairing alone is Not Enough

Forfattere: Randy Smith, Allen Parrish, Joanne Hale og David Hale. Forfatterne er tilknyttet universitetet i Alabama.

Innhold
Artikkelen refererer resultater fra en studie av parprogrammering, men kan også klassifiseres som et meningsinnlegg.
Studien ble gjennomført av forfatterne og gikk ut på å sammenligne resultater fra par som jobbet tett sammen mot par som jobbet mer individuelt. Undersøkelsen ble gjort i et profesjonelt miljø hvor oppgaven var å ferdigstille 173 moduler med ujusterte funksjonspunkter (UFP) som var definert i opprinnelig design. 48 av disse modulene inngikk i studien. Utviklerne registrerte hvilke oppgaver de jobbet med til hvilke tider, slik at man fikk oversikt over totalt tidsforbruk og grad av samarbeid på enkeltmoduler. Gjennomsnittlig antall ferdigstilte UFP pr tidsenhet utgjorde målet for produktivitet.

Påstand: Det fremsettes ingen påstand i artikkelen, kun en hypotese for studien: Tett parvis samarbeid i PP vil lede til nedsatt produktivitet og økt ressursbruk i utviklingen.

Hovedmomenter
Analyse av de innsamlede dataene fra utviklingen viser overbevisende resultater i følge forfatterne:

- par med lav grad av samarbeid ferdigstilte 4,7 UFP pr time
- par med høy grad av samarbeid ferdigstilte 1.1 UFP pr time

(UFP er gjennomsnittstall, time er produksjonstime)

De som arbeidet mest individuelt var altså over fire ganger mer produktive.
Funnene hevdes å være konsistent med tidligere studier som har vist en nedgang i produktivitet på 25-40% ved bruk av PP (dvs. høyere produksjonstid).

Konklusjon
Eksperimenter med lettvektsmetoder som innbefatter PP vil være kostbart. Utviklere må ikke tvinges til å jobbe sammen, og hvis dette blir gjort uten opplæring og tilvenning vil det være en katastrofe.
Forfatterne ønsker ikke å imøtegå de mange fordelene med PP med denne artikkelen. De påberoper seg heller ikke å ha testet prinsippene for PP fullt ut. De mener også at det i enkelte prosjekter gjerne kan være noe å vinne på produktivitet ved å bruke PP. Men denne studien viser at dette ikke oppstår av seg selv i et miljø med utviklere som ikke har trening i foreskrevne PP-prosesser.

Oppsummering
Den røde tråden mangler i denne artikkelen og det er uklart hva forfatterne egentlig ønsker å formidle. I konklusjonene er det få momenter som setter resultatene fra studien opp mot hypotesen. Det vektlegges imidlertid sterkt at det å bruke PP krever opplæring og tilvenning dersom man vil oppnå fordeler ved teknikken. Dette har trolig som formål å knytte argumenter opp mot artikkelens tittel.

Forfatterne trekker selv frem svakheter med undersøkelsen:

- ingen form for målinger i forhold til feil eller kvalitet på de ferdige modulene
- deltakerne var ikke trenet verken i PP eller lettvektsmetoder
- skiller ikke på om samarbeid har foregått på samme maskin eller etter andre samarbeidsmodeller

Videre er det store tallmessige forskjeller i funnene der det påpekes konsistens med tidligere forskning.

Undersøkelsen omfatter oppgaver som etter alt å dømme var ferdig designet. Dette krever relativt lite tankevirksomhet og aktivitet i forhold til analyse og design. Forutsatt at dette er en riktig tolkning av oppgavetypen i prosjektet, vil artikkelen ha en viss relevans for min problemstilling i den forstand at oppgavenes kompleksitet kan påvirke effekten med PP. Momentene om viktigheten av opplæring for å lykkes med PP er også av interesse. Artikkelen som helhet vil derimot ikke kunne tillegges mye vekt fordi argumentasjonen er utydelig og tvetydig. At studien er utført av forfatterne selv gjør derfor ikke så mye fra eller til.

3.3 Artikkel 3 When does a Pair Outperform two Individuals?

Forfattere: Lui, Kim Man og Keith C.C. Chan. Begge forfatterne er tilknyttet Hong Kong Polytechnic University.

Innhold
Artikkelen er en rapport fra en studie av produktivitet og kvalitet i parprogrammering. Studien er utført av forfatterne selv med profesjonelle programmerere som deltakere.

Denne artikkelen har ingen klart uttrykt påstand. Bakgrunnen for studien er å finne ut om det finnes en målbar og repeterbar test som kan påvise betydelige forskjeller i produktivitet og kvalitet mellom bruk av PP og IP. Studien baserte seg på ren problemløsning, ikke det å skrive hele program. Forfatterne mener at problemløsning er kjernen i programmering, og de ønsket å observere deltakernes evne til å løse problemer alene og i par.

Hypotesen de fremsetter er: Ved å jobbe i par forbedres evnen til problemløsning og dermed reduseres antall feil og produktiviteten øker.

Hovedmomenter
I et innledende forsøk fikk en person i oppgave å skrive samme program åtte ganger. Forsøket viste at etter å ha skrevet programmet noen få ganger er analyse, design og struktur etablert i programmererens hode, og tidsforbruket vil gå betydelig ned. At tidsforbruket reduseres etter hvert vil gjelde generelt, uavhengig av personens kompetanse og programmets kompleksitet. Dette forklares med at for hver gang det samme programmet skrives vil det kreve mindre tid til tankevirksomhet.

Resultatet fra dette forsøket vurderes opp mot to tenkte situasjoner med PP. En situasjon der et par sammen løser et helt nytt problem, en annen der de begge har løst lignende oppgave åtte ganger tidligere. Med denne vinklingen mener man å vise at det ikke alltid vil være lønnsomt å sitte to sammen, men at PP kan være mye mer produktivt når man står ovenfor helt nye utfordringer.

Studiens første del gikk ut på å måle evnen til å håndtere kompleksitet. Her ble deltakerne delt i ti enheter: fem par og fem enkeltpersoner. Deltakerne skulle løse flervalgsoppgaver basert på å analysere prosedyrer, og skulle finne riktig svar på alle oppgaver. Hele eksperimentet ble arrangert som en konkurranse hvor deltakerne fikk vite at de måtte bruke færrest mulig forsøk og minst mulig tid for å vinne. (Produksjonstid ble lagt til grunn)

Resultatet viste at parene brukte ca 20 % mer tid enn de individuelle på å levere første løsningsforslag. Når man la kvalitet til grunn, det vil si at man hadde riktig løsning på alt, brukte parene 4,2 % mindre tid enn de individuelle. Konklusjonen som trekkes er at PP er sterk på design og at PP er bedre enn individuell programmering når det gjelder designrelaterte aktiviteter.

I andre del ble det brukt en kombinasjon av å jobbe alene og i par, og her ville man måle evnen til å utarbeide en algoritme for et problem. To oppgavesett inngikk, et for individuell løsning og et for løsning i par.

Tidsforbruk for første gjennomgang var langt høyere for oppgavene som ble løst i par. Men oppgavene som ble løst individuelt hadde flere feil og det ble brukt flere forsøk og lengre tid før alt var riktig. Totalt brukte parene 5,3 % kortere tid, og de oppnådde bedre kvalitet på kortere tid. Det hevdes ut fra dette at et par har utvilsomt en større mental kapasitet siden de kan søke gjennom en større mengde av alternativer på kortere tid enn en alene kan greie.

Forfatterne refererer også et forsøk for å definere hva man i verste fall kan oppleve med parprogrammering når erfaringsnivået er ulikt i paret. Tidsforbruket ble målt på tre personer som ble kategorisert ut fra hvor raske de var: A (rask), B (middels) og C (treg).
De skrev først et program hver for seg. Deretter dannet A og C et par for å skrive et annet tilsvarende program, mens B fortsatte alene. På det andre programmet brukte A og C like mye tid som A hadde brukt alene.
Det poengteres her at selv om A kunne gjort dette like raskt alene, så har samarbeidet likevel en effekt ved at C kan forbedre sine ferdigheter gjennom kunnskapsoverføring fra A.

Konklusjon
Forsøkene viser at et par utkonkurrerer en individuell i forhold til kvalitet og produktivitet. PP utmerker seg når det gjelder å løse den type problemer som er nøkkelelementer i programmering. PP gir høyere produktivitet når programmet er utfordrende og krever mer tid til design. Et problem som er nytt for utviklerne kan løses raskere av et par fordi de kan designe en løsning raskere og bedre fordi de setter seg inn i problemet sammen og har flere alternative løsninger enn en alene vil ha. I tillegg foregår også en produktiv læring i parene.

Oppsummering
I forsøket hvor en person skrev samme program åtte ganger, viser resultater at grad og omfang av tankearbeid i programmering påvirker tidsforbruket.

Selve studien skiller seg ut fra de fleste andre ved den sterke vektleggingen på ren problemløsning. Bruk av oppgaver med fasitsvar levner ingen tvil om når alle deltakerne har oppnådd samme "kvalitet". I det første forsøket og studiet bevises to generelt aksepterte fenomen. Henholdsvis: "øvelse gjør mester" og "to hoder tenker bedre enn ett". Testene er utført med en oppgavetype som har stor relevans for PP, og funnene kan øke forståelsen for hvordan kompleksitet og analysearbeid PP påvirker produktiviteten.

Hvordan ulikt ferdighetsnivå i paret påvirker totalt tidsforbruk er en svært interessant problemstilling. Forsøket her er imidlertid gjennomført på et spinkelt grunnlag og resultatet kan derfor ikke anses som gyldig for alle situasjoner.

At studien er utført av forfatterne selv kan ha påvirket resultatene. Men en styrke ved denne studien er at de oppgavene som er brukt er tilgjengelig for alle, slik at det vil være fullt mulig å gjennomføre en identisk studie for de som måtte ønske det. Resultatene regnes som overførbare til en reell situasjon og artikkelen vurderes som svært relevant.

3.4 Artikkel 4 Pair programming & Personality Traits

Forfattere: Andrew J. Dick og Bryan Zarnett er profesjonelle systemutviklere i Red Hook Group i Canada. Red Hook Group driver systemutvikling med metoden XP.

Innhold
Artikkelen er basert på personlige erfaringer og observasjoner og er et meningsinnlegg om den menneskelige faktoren i PP. Artikkelen har to kapitler som ikke behandles her. Disse omhandler: a) viktigheten av PP som del av XP og b) intervjuteknikker for å vurdere personlige egenskaper. Jeg har vurdert disse temaene som ikke relevante for min problemstilling.

Forfatterne tar utgangspunkt i et reelt prosjekt hvor teamet besto av to seniorer (mer enn to års utviklingserfaring), fire juniorer (mindre enn ett års erfaring) samt forfatterne selv (som coach og tracker). Problemer som oppsto i prosjektet beskrives i artikkelen, og danner bakgrunn for konklusjonene.

Andrew J. Dick har via e-post gitt meg avklaring og utdyping av enkeltmomenter i artikkelen.(Vedlegg A)

Påstand: For å oppnå effektiv parprogrammering kreves det at utviklerne har spesielle personlige egenskaper.

Hovedmomenter
Det første problemet som dukket opp i det refererte prosjektet var at det ikke foregikk dynamisk rollebytte internt i parene. Dette medførte at den samme personen var driver hele tiden, mens navigatørens oppmerksomhet fløt bort til tross for stadig innblanding fra coachen. Dette forekom like mye i senior-junior-par som i junior-junior-par. Konsekvensen av en passiv navigatør var at utviklingen tok en mer komplisert vending enn det ønskede, enkleste designet.

Det andre problemet var at parenes manglende samarbeid og kommunikasjon fikk en ugunstig innvirkning på kunnskapsoverføringen i teamet som helhet, noe som igjen førte til for dårlig fremdrift. Konsekvensen ble at man valgte å gå over fra PP til IP for de to siste iterasjonene.

Dette medførte en umiddelbar økning i kommunikasjonen mellom utviklerne og dermed økt kunnskapsoverføring. En betydelig del av tiden ble også brukt til parvise diskusjoner om design og problemløsning.

En mulig forklaring på disse observasjonene i par med to juniorer er at begge antar at den andre ser helheten og vil derfor være tilbakeholden med å stille spørsmål av frykt for å virke uvitende. En aktiv navigatør er avgjørende for et vellykket design. Da de gikk over til IP ble de tvunget til å innse egen begrensning og måtte derfor stille spørsmål og kommunisere med de andre. Dette bevises ved økt kommunikasjon når PP ble avbrutt.

Forfatterne undersøkte mulige årsaker til hvorfor PP fungerte for dem selv, men ikke for de øvrige i teamet. Konklusjonen var at forfatteren har personlige egenskaper som manglet helt eller delvis hos de andre. Fire egenskaper listes opp som viktige for en utvikler som bruker PP:

Må ha evne til å kommunisere godt. God kommunikasjon er helt avgjørende i PP for å sikre seg at man hele tiden drar utviklingen i riktig og forventet retning.

Må være komfortabel med å arbeide tett sammen med andre. Konsekvensen av at man er redd for å virke dum og uvitende er at man ikke får den nødvendige flommen av ideer og forslag.

Må ha tillit til egne og partnerens evner. Trygghet i forhold til egne evner er like viktig som at man er trygg på hele teamets kompetanse. Manglende trygghet viste seg hos juniorene i vårt prosjekt ved den motviljen de viste til å foreslå endringer på andres kode.

Må evne og vilje til å inngå kompromiss selv om det betyr at egen kode eller egne forslag må forkastes. Kompromisser er påkrevet i PP og gode utviklerpar har fokus på best mulig løsning, ikke på hvem som står bak. I vårt prosjekt var det seniorene som var minst villig til å kompromisse med egne idealer, de overså sin junior-partner og reduserte dermed kunnskapsdelingen.

Konklusjon
Selv om en rekke andre egenskaper også er viktig for XP, utpeker disse fire seg som de som er nødvendig for å lykkes med PP. Disse egenskapene gjør at paret kan samarbeide effektivt for å realisere den enkleste løsningen som oppfyller alle krav.

Kommentar fra forfatteren
Artikkelen har en relativt sterk knytning til XP, og dette var bakgrunnen for at jeg tok kontakt med forfatteren. Han er enig med meg i at problematikken også har relevans for PP som frittstående teknikk. Han sier også at disse fire egenskapene er kritiske, men ikke uomtvistelige. Det vil si at det finnes unntak også her, utviklere som mangler disse egenskapene i ulik grad, kan likevel gjøre en god jobb med PP (Vedlegg A).

Oppsummering
Situasjonene som beskrives i artikkelen illustrerer problematikken godt og underbygger påstanden. Men mennesket er svært sammensatt og komplekst, og det kan være mange underliggende årsaker til problemene som oppsto. Den har også en subjektiv vinkling ut i fra at det er forfatternes personlige egenskaper som er de "rette" for PP.
Jeg mener totalt sett at det er gyldighet i den konklusjonen som trekkes med hensyn til de fire egenskapene, det er troverdig at disse er *viktige* for det tette samarbeidet som preger PP.

3.5 Artikkel 5 Is pair programming a valuable practice?

Forfatter: Daniel D. Flies er tilknyttet University of Minnesota, og artikkelen er en meta-analyse skrevet for undervisningsformål.

Innhold
Første del v artikkelen oppsummerer tidligere undersøkelser og studier av PP. I andre del argumenterer forfatteren for hvordan PP kan påvirke en bedrifts Software CMM og People CMM. I omtalen av CMM og KPA har jeg valgt å bruke begrepene på originalspråket.

Formålet med artikkelen er å undersøke hvilke fordeler PP kan medføre for en organisasjon og dens ansatte, og for programvaren som de utvikler.

Refererte funn
Innledningsvis refererer forfatteren en del hovedlinjer om fordelene med PP basert på funn gjort i tidligere studier:

PP ser ut til å gjøre kommunikasjon og kunnskapsspredning til en integrert del av hver kodelinje som skrives. Kunnskap om systemet desentraliseres, fordi minimum to personer er ansvarlig for hver kodelinje. Utviklerne deltar dessuten i løsning av alle typer oppgaver som følge av rullering. Dermed styrkes også utviklerne ved at de bygger opp bred kompetanse.

PP gjør det lettere å avdekke og forebygge feil fordi det er to par øyne som kontrollerer. Det vises her til Delphi-effekten som er en term skapt av sosiologer: Meningen fra en gruppe av mennesker med likeverdig kunnskap er mer pålitelig enn meningen fra et tilfeldig valgt enkeltindivid.

Forfatteren sammenligner videre resultater fra to tidligere studier, den ene utført av John T. Nosek [5], den andre av Jerzy Nawrocki og Adam Wojchiechowsky [16]. Ut fra disse funnene påpeker han at effektiviteten ved PP ikke kan måles bare ut fra LOC per time, man må også se på programmets funksjonalitet og lesbarhet, fordi det er her PPs styrke ligger.

I andre del av artikkelen tar forfatteren for seg hvordan PP kan hjelpe en organisasjon til å oppnå et høyere nivå i Software-CMM. Han mener at en rekke Key Process Areas (KPA) kan påvirkes så å si umiddelbart:

> *Developer efficiency*: studier har vist at gjennomsnittlig utviklingstid er lavere med PP enn med IP.
>
> *Peer Review*: PP er mer effektiv enn andre teknikker på dette området fordi det her også inngår et forebyggende aspekt.
>
> *Defect prevention*: I PP har hver utvikler bedre oversikt, og ser lettere de mønstrene i helheten som medfører at defekter oppstår. Koden har god lesbarhet og er lettere å vedlikeholde.
>
> *Software product engineering*: Dette KPA vektlegger bla forutsigbar utviklingstid, og PP er mer effektiv enn IP når det gjelder å utvikle kode på en forutsigbar måte.

Training program: Vanskelig å foreta kvantitative studier på dette området, men det finnes anekdotiske bevis som går på at opplæring kan integreres i det daglige gjennom PP uten at det går ut over verdifull utviklingstid eller personlig fritid.

I People-CMM mener forfatteren at PP kan gi fordel på følgende levels og KPA:

Level 2: *Work environment, training and commuication.* PP innebærer "peer pressure" som gir et mer fokusert utviklingsmiljø.

Level 3: *Participatory culture* og *knowledge and skills analysis.* PP gir bedre oversikt over den enkeltes sterke og svake sider, og fremmer samarbeid.

Level 4: *Team-based practices, team building and mentoring*. PP er i seg selv teamarbeid og vil medvirke til mer samlet organisasjon. Rådgivning og hjelp skjer ofte i en uformell sammenheng i PP, mens CMM krever en mer formell form. Men kunnskapsoverføringen i PP er likevel viktig.

Level 5: *Continous work-force innovation, coaching* og *personal competency development.* PP hjelper organisasjonen gjennom alle de fire tidligere nivåene for å nå nivå 5.

Konklusjon
Forfatteren konkluderer med at svaret på det innledende spørsmålet er ”ja”; PP medfører en rekke fordeler. Ved å bruke PP på alle utviklingsaktiviteter vil man oppnå fordel på flere områder. Fra individuell kompetanseheving til at organisasjonen leverer produkter med høyere kvalitet og øker verdien på sin viktigste ressurs: de ansatte.

Oppsummering
Kunnskapsspredning, færre feil og høyere produktivitet er de viktigste fordelene forfatteren trekker frem. Han fremhever "dobbeltsjekkingen" – fire øyne ser bedre enn to - som den mest sentrale årsaken til at PP gir kvalitetskode. Disse momentene anses å ha rimelig gyldighet.

En organisasjon består oftest av flere profesjoner enn programmerere, og også flere prosesser enn det som omfattes av PP. Det er derfor verdt å merke seg at forfatteren presiserer at PP *kan være til hjelp* for å nå høyere nivåer i CMM. En vurdering av hvorvidt momentene har gyldighet som godkjenningskriterier i CMM vil kreve en dypere analyse. Dette er ikke foretatt her, fordi dette går utenfor min problemstilling.

Første halvdel av artikkelen er relevant for min problemstilling, men med det forbehold at artikkelen i hovedsak er en beskrivende oppsummering av tidligere studier.
Andre del er også relevant ved at fordeler ved PP listes opp og settes i et perspektiv mot en kjent modell for prosessforbedring.

3.6 Artikkel 6 Paired Programming in the Software Factory

Forfattere: Richard Sheridan, Thomas Meloche og James Goebel. Forfatterne er henholdsvis president og avdelingssjefer ved Menlo Institute, Michigan USA. Selskapet driver systemutvikling i The Menlo Software Factory, og også kurs- og foredragsvirksomhet. Firmaet benytter XP som utviklingsmetode.

Innhold
Artikkelen besvarer ofte stilte spørsmål om PP, og er rettet mot organisasjoner som ønsker å prøve teknikken. Artikkelen betoner sterkt at utvikling med PP gir høykvalitets-produkter, og den kan derfor sies å et kommersielt formål gjennom at man indirekte markedsfører bedriftens tjenester og kurstilbud. Men den er i første rekke å betrakte som en erfaringsrapport.

Hovedmomenter
Tema for første del er: Hvorfor er produktiviteten høyere med PP enn med IP? Hvor ligger "kompensasjonen" som resulterer i høyere produktivitet? Forfatterne mener at produktivitet ikke alene kan måles i LOC, funksjonspunkter eller lignende. De beskriver produktivitet med spørsmålet:

'*How much customer value can we create per dollar spent?'*

Momenter og argumentasjon for at produktiviteten øker med PP:

Økt kvalitet på resultatet. To personer er hele tiden fokusert på og opptatt av kvaliteten på resultatet, og sannsynligheten for å avdekke feil øker. Produktiviteten vil påvirkes positivt selv om paret bare finner én ekstra feil pr uke, fordi kostnadene blir langt større dersom feilen oppdages senere.

Mer effektiv arbeidstid. Man vil ikke så lett bli låst av ett problem fordi man har en partner som aktivt vil bidra hele tiden for å løse problemet. Ved IP sitter man ofte i timevis fordi det å spørre om hjelp anses som et svakhetstegn. Når det jobbes i par reduseres også de ikke-produktive aktiviteter, fordi man nødig gjør dette så lenge man har en partner med seg.

Alltid to perspektiver på enhver løsning. Her vektlegges verdien av å ha en blanding av seniorer og juniorer, både med hensyn på kunnskapsutveksling og at man har ulike perspektiver. Seniorene har erfaringen, juniorene har frisk kunnskap om nyere teknologier og nye måter å løse problemer på.

En atmosfære av hjelpsomhet. Teamet er positivt preget av at barrieren for å be om hjelp er brutt. Alle er villig til å hjelpe enhver som spør og alle er villige til å spørre om hjelp. Langt mindre redsel for det ukjente. Et par er langt mer villig til å fordype seg i ny og ukjent teknologi enn en utvikler alene er.

Koden blir ikke større enn nødvendig. Det kan være et problem for en utvikler å avslutte koden når oppgaven egentlig er løst fordi han ikke "ser" det. Erfaringen her er at dette sjelden skjer i et par fordi partneren ofte vil stille spørsmål til utvidelsen.

Tema for andre del av artikkelen er parsammensetning. Bedriften har valgt å styre dette ved at en leder setter sammen par for hver iterasjon. Man vektlegger å sette sammen par ut fra å kombinere talent, erfaring og ferdigheter, samtidig skal man ivareta kommunikasjonen og kunnskapsspredningen i teamet. Utviklerne får også god erfaring med å forholde seg til nye partnere, noe som er gunstig for effekten av tilfeldig arbeidskraft som leies inn i perioder.

Junior-seniorpar: Det tilbakevises at et slikt par kun dreier seg om at senior lærer opp junior. Kunnskapsdeling skjer nesten alltid begge veier. Juniorer og middels erfarne utviklere er svært viktige for teamet, det stimulerer til nye ideer og sikrer at koden kan vedlikeholdes av juniorene i fremtiden.

Junior-juniorpar: Bedriften har trosset råd om ikke å sette to juniorer sammen. To juniorer er ikke like produktive som et par med minst en senior, men det interessante er at det er mulig med denne sammensetningen. Det er en fordel om begge har vært i par sammen med noen andre først, så når nye utviklere kommer til teamet settes disse alltid først sammen med en som er erfaren med prosessen og verktøyene.

Senior-seniorpar: Artikkelen har ingen momenter som går direkte på denne partypen, men trekker frem at mange seniorer kan ha tendens til å:

- skrive kode som er for kompleks for andre å forstå eller vedlikeholde
- overdrive løsningen og produsere mer enn nødvendig
- å bruke tilnærminger som har fungert tidligere, men som ikke lenger er effektive

Ut fra dette anbefales det at minst halvparten av utviklerne i teamet er på junior- eller middelnivå, et team av bare seniorer er ikke et "dream-team".

Konklusjon
Forfatterne hevder at PP har ført med seg betydelige fordeler for denne bedriften som langt på veg har overgått kostnadene som er forbundet med å la to programmerere jobbe med samme kode.

Oppsummering
Artikkelen som helhet gir en svært positiv og tilnærmet ukritisk anbefaling av PP, og i argumentasjonen settes enkelte forhold på spissen. Den positive vinklingen kan skyldes at man gjennom artikkelen også ønsker å markedsføre bedriftens tjenester, både kurs og foredrag og utvikling av det som forfatterne refererer til som ”Killer applications”. Med dette som en noe dempende faktor vil jeg anse at argumentasjonen har relativt høy gyldighet.

3.7 Artikkel 7 Conversant Pairing

Forfatter: Nathaniel Talbott er profesjonell konsulent innenfor systemutvikling med mer enn ti års erfaring.

Innhold
Artikkelen beskriver en teknikk for å styre interaksjonen når man programmerer sammen med en annen person. Teknikken er basert på egne erfaringer.

Innledningsvis hevder forfatteren at dersom paring ikke gjør deg mer produktiv og mer glad og fornøyd, så er det fordi du ikke gjør det riktig.
Riktig paring er ikke som å lese en tegneserie over skuldra på noen, men mer som å ha en interessant konversasjon på en fantastisk fest, sier han. For å skille riktig paring fra "dobbeltgjengere" bruker han begrepet "Conversant Pairing", fordi han mener at konversasjonen – det å føre en dialog – er nøkkelen til suksessfull paring. Han presenterer ett sett med prinsipper som utgjør hans modell for vellykket paring.

Prinsipper
Spørsmål er en konstant del av en god PP-økt. En muntlig forklaring kan ofte føre til ny forståelse og løsning på et problem, og man får feedback på egne tanker.

Interesse og engasjement for oppgaven fra begge parter er en kritisk faktor. Selv på den kjedeligste oppgave må du om ikke annet *spille* interessert. Dette vil ofte lede til ekte interesse.

Hastverk er lastverk. Bruk tid på spørsmål og diskusjoner for å finne den enkleste løsningen. Har man det for travelt kan man få unødvendig mye kompleksitet. Men ikke overdriv dette ved å føre lange diskusjoner på hver minste detalj.

Kontrovers leder til gode diskusjoner. Ta opp ting til diskusjon og vær ikke redd for uenighet. Sørg for å bevare en lett tone, bygg opp trygghet og vis litt ydmykhet.

Relasjonsbygging er viktig for samarbeidsklimaet i paret. Bli kjent med partneren for å kunne føre en bredere dialog.

Artikkelen inneholder også et kapittel om hva som kan være til hinder for en vellykket paring. Her presenteres typiske problemsituasjoner med forslag til hvordan man kan håndtere disse.

Svært ulik erfaring kan medføre at dialogen uteblir helt eller delvis. Forfatteren hevder at dette skyldes stolthet. En junior er for stolt til å ta sjansen på at hans spørsmål og forslag er for dumme. Seniorens stolthet går på at han ikke har bruk for innspill og forslag fra en uerfaren junior. Forfatterens råd til junior er at han må innse at han aldri vil lære noe hvis han ikke stiller spørsmål og kommer med forslag. Senior må huske at det å snakke om hva han gjør hjelper til å reorganisere tankene og at det er mulig å bli overrasket av et innsiktsfullt forslag fra junior.

Argumentasjon. En diskusjon som handler om å finne den beste løsningen kan gå så langt at den ender opp i en argumentasjon der målet kun er å vinne. En diskusjon ender som regel med at begge sider har kompromisset og man har en løsning som er bedre en noen av partene ville greid på egenhånd. Dersom man merker at en diskusjon begynner å nærme seg argumentasjon bør en tredje part trekkes inn for å si sin mening.

Asosial oppførsel er en omfattende type hindring. Dette kan være å vise motvilje mot paring, snakke hele tiden, ikke snakke i det hele tatt, gjøre narr av andre, dårlig hygiene og så videre. Det som er viktig å være oppmerksom på her er at den som oppfattes som asosial kanskje ikke oppfatter seg selv som det, eller er klar over det. Man bør si fra, men også her vise ydmykhet.

Forfatteren konkluderer med at noe av det beste med PP er kunnskapsoverføringen, det at man aldri slutter å lære. Hans egen erfaring er at hver eneste person han har vært i par med, har lært ham noe nytt hver gang de har samarbeidet.

Oppsummering
Artikkelen beskriver troverdige situasjoner fra et PP-miljø og jeg anser den som en svært interessant erfaringsrapport. At en levende dialog er essensiell i PP er godt underbygget med egne erfaringer og anekdotiske betraktninger som man kjenner seg igjen i.

3.8 Artikkel 8 Ekstrem Programmering – Praktiske erfaringer

Forfatter: Trond Johansen, leder for forskning og utvikling i firmaet Proxycom AS. Proxycom driver i hovedsak med systemutvikling og IT-rådgivning.

Innhold
Proxycom ønsket å prøve ut XP i et prosjekt med bakgrunn i alle fordeler de hadde sett beskrevet med metoden. De valgte seg et internt prosjekt for forsøket, hvor prosjektgruppen besto av ni personer (inkludert en kunde). Artikkelen er en erfaringsrapport fra dette prosjektet. Johansen utarbeidet også en norsk veiledning for XP i forbindelse med prosjektet for å forberede prosjektmedlemmene [13].

I artikkelen beskrives positive og negative erfaringer fra prosjektet knyttet til disiplinene i XP, og forslag til forbedringer av XP. I denne vurderingen vil kun momenter som angår PP belyses fordi det øvrige ikke anses å ha direkte relevans for min problemstilling. Momentene er hentet fra konklusjonen i artikkelen.

Positive erfaringer

- PP var effektiv når en utvikler skulle opplæres
- PP er mer effektiv ved utvikling av komplekse systemer, uklare krav og korte frister
- Kvaliteten på programkoden ble bedre eller like bra som med tradisjonelle metoder.
- Felleseie av programkode er fordelaktig
- Det antas at man får en form for kodelesingseffekt med parprogrammering

Negative erfaringer

- 50 % høyere tidsforbruk med XP-metoden enn med tradisjonelle metoder.
- Ser ut til at PP ikke er så effektiv som det hevdes. Spesielt har vi sett liten nytte i parprogrammering av enkle og trivielle oppgaver.
- Vanskelig å danne par når deltakerne også jobber på andre prosjekter.

Forslag til forbedringer
Ut fra disse erfaringene foreslås to tiltak for å fjerne eller redusere de negative sidene med metoden XP. Begge tiltakene er rettet mot PP:

> *Redusere bruken av PP*. Det foreslås at PP brukes ved opplæring og for komplekse programmoduler, hvor utviklerne selv avgjør hva som bør gjøres i par.
>
> *Utviklerne bør jobbe heltid på prosjektet*: Dette vil eliminere problemet med at man ikke finner en partner for PP.

Oppsummering
Konklusjonene er trukket ut fra svarene man fikk i en spørreundersøkelse i prosjektgruppen. Spørsmålene som ble stilt var helt åpne, og det ble dermed mange ulike svar og formuleringer som kan gi rom for tolkning.
Erfaringene bygger på bare ett prosjekt er et dårlig grunnlag for å bedømme metoden XP som helhet. Når det gjelder PP vil grunnlaget (antall oppgaver) være på høyde med flere av de andre artiklene som er vurdert i dette kapittelet. I dette prosjektet var det imidlertid ikke bare PP som var nytt for utviklerne, men også hele metoden rundt. Dette kan ha påvirket resultatene, ved at positive og negative effekter var sammensatt av flere årsaker. På grunn av disse forholdene anses momentene å ha liten generell gyldighet for PP.

3.9 Artikkel 9 Praktiske erfaringer fra programmerere

I dette punktet oppsummeres resultatet av en spørreundersøkelse jeg har gjennomført mot tre profesjonelle programmerere. For ikke å komme i tidsnød med håndtering av svarene, ble spørsmålene utarbeidet i begynnelsen av april, før skrivearbeidet hadde kommet skikkelig i gang. Jeg konsentrerte meg om å dekke de punktene knyttet til PP som frem til da syntes relevante i forhold til den foreløpige problemstillingen og det jeg hadde lest.

Spørsmålene ble utformet med tanke på å unngå misforståelser (også språklige), og har delvis form som problemstillinger. Problemstillinger og spørsmål er gjengitt i sin helhet i vedlegg B, hvor også de fullstendige svarene er lagt inn. Respondentene benevnes som A, B og C.

A, B og C, har henholdsvis 4, 3 og 1 års erfaring med PP i tillegg til generell erfaring fra programmering (jfr. vedlegg B). Alle har i hovedsak brukt PP som en integrert del av XP, men A har i tillegg praktisert PP innenfor andre metoder.

Spørsmålene er sortert ut fra tema, og gjengis i en kortform her. Spørsmålene med innledning er uthevet i kursiv. For å lette lesingen er svarene fra de enkelte respondentene delvis sammenstilt.

Fordeler med PP
Her ble fem punkter presentert som ofte refererte fordeler med PP, her nummerert og noe forkortet. To spørsmål ble stilt her:

'Er dette fordeler du selv har opplevd?' og 'Har du opplevd andre fordeler med PP?'

1) *Høyere kvalitet*: B og C bekrefter at de har opplevd dette. A antar at PP alene kan bidra til et bedre design og dermed bedre kvalitet. Han mener at han ikke har grunnlag for å uttale seg om dette siden han praktiserer testdrevet utvikling og mener dette er hovedårsaken til bedret kvalitet.

2) *Gladere og mer fornøyde programmerere*: A og B har begge erfart dette, men A mener at dette avhenger av kjemien mellom utviklerne, og at begge er aktive i arbeidet. C mener at dette først oppstår når teamet er innkjørt.

3) *Kunnskapsspredning*: Alle har opplevd dette, men A påpeker at det forutsetter at man rullerer parene.

4) *Kortere utviklingstid*: A og B sier seg helt enig i dette og A fremhever at noe av årsaken er at man arbeider mer intensivt i par. C mener at det også her kreves en innkjøringstid, men han erfarte i sitt (korte) prosjekt at hastigheten steg etterhvert.

5) *Færre feil i sluttproduktet*: A viser også her til testdrevet utvikling, mens B sier seg enig. C viser til sitt svar om andre fordeler.

Andre fordeler: A og B trekker frem høy grad av samhold og hjelpsomhet i teamet, og det at alle arbeider for helhetlig kvalitet og ikke bare deler. C fremhever den integrerte kodegjennomgangen som største fordel, og tror styrken ligger i at den skjer kontinuerlig og ikke i ettertid.

Kritiske suksessfaktorer
Dette punktet omfatter spørsmål om personlige egenskaper og parsammensetning.

Personlige egenskaper: Utgangspunkt for dette punktet er de spesifiserte personlige egenskapene fra artikkel 4: Kommunikasjon – Kompromiss – Samarbeidsevne – Trygghet (i forhold til egen kode).

'Er du enig i at disse egenskapene er kritiske suksessfaktorer?' og 'Er det eventuelt andre egenskaper/ menneskelige forhold du anser som like viktige/ viktigere?'.

A er enig, men man må ikke dra forhastede slutninger om egenskaper - la folk få prøve PP. Trygghet på egne evner må modnes. Viktig også å vise åpenhet og ikke være forutinntatt om andres kode. B er også enig at dette er viktig, men hevder at PP i seg selv (med en god partner) kan gjøre deg sterkere på alle disse egenskapene. Han tror at PP kan bli svært vanskelig for personer som har problemer på flere av de fire områdene. C mener at det viktigste er at de to i paret er på samme teknologiske nivå.

Parsammensetning
Rullering: temaet går på erfaringer med ulike måter å rullere parene på, hva erfaringene er og hva som eventuelt anbefales.

A og B sier at man bør velge partner selv, men styrt rullering bør brukes hvis sammensetningene blir veldig statiske eller når det er utviklere i teamet som er uerfarne med PP. C har brukt styrt rullering med bytte for hver iterasjon, men har opplevd problemer på grunn av spredt teknisk kunnskap i teamet som satte begrensninger for rullering. Ser viktigheten av at rullering øker kunnskapsoverføringen, men opplevde et press om ikke å rullere for mye av budsjetthensyn.

Rollebytte: Her spørres det om viktigheten av at hver partner har sin bestemte rolle, og i hvilken grad bør kunnskap/ erfaring legges til grunn?

Alle mener at man i utgangspunktet må bytte rolle etter som det passer, blant annet på grunn av intensiteten i PP (A). Erfaring bør ikke være avgjørende, det som er viktig er at juniorer oppmuntres til å hevde sine meninger (A), og at junior må få være driver hvis det er vanskelig å henge med som navigatør (C). C mener at balanserte par er mest effektive, stort sprik i erfaringsnivå fungerer best i opplæringssituasjoner.

Andre kritiske faktorer
Her ble deltakerne spurt om de ser andre kritiske faktorer enn personlige egenskaper og parsammensetning. Følgende punkter trekkes frem:

- bedriftskulturen må være preget av samarbeid på alle plan
- mer effektive møter
- best med utviklingsmetode hvor blant annet design og implementasjon går parallelt, og krav til formell design ikke er så strenge
- gjensidig tillit i hele teamet
- teamet må ha full forståelse for hva PP innebærer
- fysisk arbeidsmiljø bør være slik at det er lett å bytte arbeidsplass (kontorlandskap)
- lavt støynivå

Individuelt arbeid
Dette punktet går på om PP bør brukes på all kode, eller om man også bør benytte IP.

A og B foretrekker PP fullt ut, men IP brukes for enkelte oppgaver fordi det er smidigere eller fordi det er ulikt antall utviklere til stede. IP krever gjennomgang av paret i ettertid og dermed er det ikke tidsbesparende. Bør uansett være enighet om hvilket nivå man skal drive PP på (jfr ulike arbeidstider.) Ikke alene for lenge om gangen.
C mener at enkel kode som kan skrives uten å tenke, bør genereres. Det meste av koden blir fort så komplisert at det er hensiktsmessig å bruke PP.

Oppsummering
Som beskrevet i punkt 1.1 brukte jeg begrepet *kritiske suksessfaktorer* i spørreskjemaet. Da jeg fikk tilbake svarene så jeg ut fra variasjonen i svarene at begrepet tolkes svært vidt. Etter min forstand er for eksempel støynivå *ikke* kritisk. Momentene som kommer frem her er likevel av stor interesse og jeg valgte å ta disse med videre som *viktige forutsetninger*.

Alle respondentene har i hovedsak sin erfaring med PP i sammenheng med XP. Dette vil erfaringene være farget av i en viss grad, men mange av svarene speiler en uavhengig og personlig mening.

Synspunktene fra A og B vil ha størst gyldighet av de tre, siden disse har lengst erfaring. Men svarene fra C er også av interesse, ikke minst fordi de indikerer at en del forhold ved PP først viser seg etter en viss tid.

4 Funn i kildene vs problemstilling

Kildene som er vurdert i kapittel 3 vil i dette kapittelet bli diskutert opp mot problemstillingen for oppgaven:

> *Hvilke fordeler kan man oppnå ved å bruke parprogrammering framfor individuell programmering, og hva er kritiske suksessfaktorer?*

Jeg diskuterer først fordeler, deretter viktige forutsetninger. Artiklene i kapittel 3 vil i dette kapittelet bli referert ut fra nummeret de er tildelt, i tillegg til ordinær henvisning til litteraturlista.
I tillegg til artiklene fra kapittel 3, vil det også i noen tilfeller refereres til andre kilder.

4.1 Rammeverk

I diskusjonsprosessen har jeg brukt momenter fra et foreslått rammeverk for forskning på PP [14]. Dette er utarbeidet for å støtte empiriske studier og meta-analyser av PP, slik at man lettere skal kunne sammenligne resultater fra ulike kilder. Rammeverket forsøker å identifisere og kategorisere viktige variabler knyttet til studier av PP, og å gi en grunnleggende forståelse for forholdet mellom disse variablene (ibid).

De variablene jeg anser som viktige i forhold til denne oppgaven, vil bli omtalt fortløpende der de er lagt til grunn eller har relevans for diskusjonen.

4.2 Fordeler med parprogrammering

I dette punktet behandles de fordelene ved PP som fremkommer av artiklene i kapittel 3. I henhold til rammeverket [14] er det viktig å knytte fordelene ved PP til målbare variabler, i den grad det er mulig. Variablene kan betraktes som resultatet vi ønsker å oppnå ved å bruke PP (ibid).

4.2.1 Bedre kvalitet

I de fleste artiklene er kvalitet målt ut fra antall feil og dette kan regnes som hovedessensen i bedret kvalitet. Antall feil i koden vil påvirke arbeid med gjenfinning og retting av feil, og det vil påvirke funksjonaliteten til produktet som leveres til kunden. Design fremheves også av flere som et kvalitetsmål. Dette er viktig i forhold til enklere implementasjon, gjenbruk og vedlikehold. I dette punktet diskuteres hvorfor PP gir bedre kvalitet.

Flere artikler fremhever at design og kode blir bedre fordi to perspektiver gir flere alternativer og fordi to personer til enhver tid er fokusert på kvaliteten i resultatet [4][6][9][11]. Den kontinuerlige kontrollen under implementasjonen medfører at et stort antall feil forhindres helt fra starten av utviklingen [4]. Artikkel 5 [10] sier også at det er lettere å avdekke og forebygge feil når det er to par øyne som kontrollerer, og knytter dette til Linus' lov:

"Given enough eyeballs, all bugs are shallow"

Kontinuerlig kodegjennomgang er en viktig styrke ved PP, og utmerker seg fordi det er en teknikk som faktisk fungerer i motsetning til mange andre [11] (vedlegg C).

Hovedmomentet for at PP gir bedre kvalitet er at man alltid har to perspektiver og kontinuerlig kodegjennomgang. Dette er fakta som det ikke er noen grunn til å tilbakevise. Den effekten dette gir er godt underbygget gjennom undersøkelser, studier og erfaringer. Bedre kvalitet vil også kunne gi avledede fordeler i form av reduserte kostnader til retting av feil og vedlikehold av koden.

4.2.2 Kortere leveringstid

Kortere leveringstid er en betegnelse jeg har valgt å bruke for forhold som ofte betegnes som "økt produktivitet". Produktivitet er et vidt begrep, og er ikke alltid like godt definert. Rammeverket [14] anbefaler også at variabelen tid baseres på medgått tid. Begrunnelsen for dette er at kunden er mest interessert i *når* produktet kan leveres, og at leveringstidspunket er stabilt og itl å stole på.

I forbindelse med produktivitet er LOC ofte brukt som et mål på produktivitet. Sheridan et al [11] advarer mot dette. Det er *ikke* slik at utvikleren har all kode i hodet og at produktivitet bare er snakk om hvor raskt han klarer å skrive på tastaturet (ibid). Antall kodelinjer i det ferdige produktet blir dessuten ofte færre med PP enn med IP. Færre kodelinjer er en indikasjon på at PP gir bedre design [16]. Disse momentene baserer seg på resultatene fra den studien som er omtalt i artikkel 1 [4].

I artikkel 3 ble det konstatert at et par har en større mental kapasitet siden de kan søke gjennom en større mengde av alternativer på kortere tid enn en alene kan greie [9]. Dette støttes også av artiklene 5 og 6 [10][11].

PP har stor effekt på analyse- og designaktiviteter, og kompleksiteten i løsningen eller oppgavetype vil være en viktig variabel her [4][8][9][12](vedlegg C). Jo mer kompleks oppgave, jo større blir forskjellen i medgått tid mellom PP og IP [9]. Ved PP treffes det raskere beslutninger i forhold til hva man skal implementere, fordi to hoder tenker bedre enn ett [4] [9].

Talbott sier at PP er en ekstremt interaktiv prosess som kan bli svært intens [6]. Dette støttes også i spørreundersøkelsen (vedlegg C). Talbott fortsetter med å si at intensiteten i perioder medfører en høyere produktivitet på kortere tid [6].

Nawrocki og Wojciechowski viser gjennom sine studier indikasjoner på at PP er mer effektiv enn IP i forhold til å produsere programkode på en forutsigbar måte [16]. Noseks studie støtter også at PP gir mer konstant utviklingstid [5].

Momentene om at LOC ikke er et godt produktivitetsmål er troverdige fordi det bygger på reelle undersøkelser og erfaringer. Det bekrefter også påstanden om at PP gir et bedre design, fordi et gjennomtenkt design gjør det letter å produsere kort og konsis kode. Dessuten er det å finne den enkleste løsningen med *færrest* mulig kodelinjer et implisitt mål i PP.

Argumentasjonen for at PP leder til kortere leveringstid er solid og pålitelig, men det er viktig å se dette i sammenheng med kompleksiteten i koden (jfr 4.4.2).

4.2.3 Økt jobbtilfredshet

Dette punktet tar for seg påstander knyttet til at utviklere som praktiserer PP er mer glade og fornøyde og føler større sikkerhet på egen kode og løsninger.

At programmerere blir mer glade og fornøyde med PP er et utsagn som står sentralt som en del av konklusjonen i artikkel 1 [4]. Dette bygger på direkte innspill fra studenter og profesjonelle. Noen av årsakene til økt jobbtilfredshet refereres i artikkelen:

- større trygghet på eget arbeid
- alltid noen som kan hjelpe hvis du står fast
- bedre mulighet til å få luftet ideer
- mer tid på utfordrende oppgaver og mindre tid på irriterende feilsøking
- å løse en oppgave sammen med andre gir en oppstemt følelse ”vi greide det”

Talbott [6] påpeker at hvis du ikke trives bedre med PP enn med IP er det fordi du ikke gjør det riktig. Han mener at det aller beste med PP er at du hele tiden lærer noe nytt.

I spørreundersøkelsen sier en av respondentene at det blir bedre samhold i teamet, alle hjelper hverandre og alle jobber for helheten – ikke bare for deler av systemet (vedlegg C).

På internett kan man i faglige diskusjonsforum og lignende finne en rekke meningsinnlegg som indikerer at mange utviklere trives bedre med PP enn med IP, uten at jeg kan gå god for validiteten i disse uttalelsene. Om man opplever økt trivsel med PP vil trolig også avhenge av personligheten, utadvendte personer vil kanskje trives best med PP [3].
Dette er et område som vanskelig lar seg måle annet enn gjennom intervjuer, og alle synspunkter vil da også være subjektive. Dermed vil man ikke kunne trekke en konklusjon, men behandle momentene som indikerende tendenser.

4.2.4 Kunnskapsoverføring

Rammeverket sier følgende om variabelen som dekker kunnskapsoverføring:
Informasjon og kunnskap kan være både taus og eksplisitt. PP kan være godt egnet til å dele stilltiende kunnskap gjennom at man i paret observerer hverandres arbeid [14].

Ingen av de vurderte artiklene har foretatt noen form for måling eller systematisk oppsummering i forhold til kunnskapsoverføring, så det vil ikke kunne utføres en sammenligning i så måte. Men flere artikler fremhever kunnskapsoverføring som en betydelig styrke ved PP [7][9][10][12](vedlegg C), og flere fordeler er knyttet til kunnskapsoverføringen som er integrert i PP:

- systemspesifikk kunnskap er spredt i hele teamet, og det er derfor mindre risiko for problemer dersom noen slutter eller blir borte fra prosjektet
- kompetanse som ressurs i bedriften forvaltes bedre gjennom at den er spredt på mange
- utviklere tilegner seg ny kunnskap gjennom at de lærer av hverandre, og får erfaringer fra et bredt fagfelt
- enklere vedlikehold fordi flere kjenner koden
- bedriftens kostnader til kurs og opplæring reduseres

Kunnskap er av mange fremhevet som den viktigste ressursen i en bedrift og omtales ofte som intellektuell kapital. I en artikkel om kunnskapsdeling [18] sies det at en bedrift som vektlegger kunnskapsforvaltning vil ha store muligheter til å lykkes. Overføring av kunnskap og erfaringer fra individet til organisasjonen trekkes frem som et viktig element her, og videre organisasjonens evne til å gjøre denne kunnskapen til sin (ibid).
Kunnskapsoverføringen i PP kan vise seg å bli en svært viktig fordel for bedriften hvis man skal tro disse påstandene.

4.3 Menneskelige faktorer

Dette er et sentralt og komplekst område innenfor PP og det er derfor behandlet i et eget underkapittel. I dette punktet diskuteres parsammensetning i forhold til erfaringsnivå. Personlige egenskaper diskuteres i et eget punkt da dette ikke er satt i direkte sammenheng med parsammensetning i noen av kildene.

4.3.1 Parsammensetning

Problematikken ved parsammensetning er i første rekke knyttet til PP-bakgrunn, erfaringsnivå og ferdigheter. I følge Williams og Kessler [3] stilles det også ofte spørsmål om hvordan kjønn og kulturell bakgrunn kan påvirke samarbeidet i paret, men de har erfart at dette er momenter som har liten eller ingen betydning i PP. For å beskrive ulikt erfaringsnivå brukes junior- og seniorbegrepet (jfr punkt 1.3).

Junior-junior er den parsammensetningen som det er knyttet mest skepsis til, og som mange vil prøve å unngå [11]. En viktig forutsetning for denne typen par er at begge har noe erfaring fra PP slik at de har kunnskap om prosessen og verktøyene (ibid)[3].

I prosjektet som er referert i artikkel 4, observerte man at junior-junior-par kommuniserte svært dårlig. Man antok at dette i stor grad skyldtes at begge var redde for å virke uvitende og derfor unnlot å stille spørsmål [7]. Talbott har også erfart tilsvarende tilbakeholdenhet hos juniorutviklere [6]. Dette betyr at kommunikasjonen kan bli berørt også i andre

konstellasjoner der en ”taus” junior inngår. Dette bekreftes også av en respondent i spørreundersøkelsen (vedlegg C).

Williams og Kessler [3] påpeker at siden juniorer per definisjon mangler erfaring, kan det være fare for at junior-junior-par blir ineffektive. Dette på grunn av at de velger en unødvendig komplisert løsning, beveger seg bort fra overordnet design og/eller de bruker veldig mye tid på enkle ting.

En fremtredende styrke hos juniorer er at de gjerne har kunnskaper om ny teknologi og nye tilnærminger som kan overføres til seniorer og gi ny inspirasjon [11].

Junior-senior-par oppfattes av mange som en situasjon hvor senior lærer opp junior, men det er viktig å være klar at det ikke skal være slik [11]. I dette paret er det svært viktig at senior oppmuntrer junior til å komme med innspill, og bygger opp tillit fordi juniorer kan ha svært gode forslag (Vedlegg C). Seniorer kan ha problemer med å ta råd og innspill fra en junior [6][7]. En konsekvens av dette er redusert kunnskapsoverføring.

Senior-senior-par er den parsammensetningen hvor det er mest trolig at det oppstår ego-relaterte konflikter [3]. Seniorer kan ha problemer med å forkaste egne ideer (ibid), på grunn av et sterkt behov for å eksponere sin ekspertise. I artikkel 4 erfarte de også dette; seniorer var minst villig til å kompromisse med egne idealer [7].
Den største styrken i et slikt par er en økt evne til å løse ekstremt komplekse problemer [3].

En av forfatterne av artikkel 6 [11], Thomas Meloche, sier i en tilleggskommentar at riktig parsammensetning er viktigst i et langsiktig perspektiv, og man skal ikke bekymre seg for om man har satt sammen det optimale paret for en enkelt iterasjon (vedlegg A).

4.3.2 Personlige egenskaper

Rammeverket [14] poengterer at en viktig utfordring for en leder er å finne ut av hvilke personlighetstyper som inngår, eller burde inngå i teamet. PP er en teknikk med en utpreget sosial profil, og det finnes mange ulike egenskaper som på hver sin måte kan påvirke effekten. I hovedsak er det er gjerne hvorvidt personen er utadvendt eller innadvendt som er av størst interesse i PP-sammenheng (ibid).

Artikkel 4 konkluderer med at fire egenskaper må være til stede for å lykkes med PP. Dette dreier seg om evne til å kommunisere, kompromisse, samhandle med andre og å være trygg på egne (og partnerens) evner [7]. I en utfyllende kommentar sier forfatteren (Dick) at det finnes unntak, utviklere kan fungere godt i PP selv om en eller flere av disse egenskapene mangler. Han mener også at utadvendte personer egner seg spesielt godt for PP. Han føyer også til at det er en fordel om man ser på en PP-økt som en mulighet til å lære noe nytt eller forbedre seg (vedlegg A).

De fire egenskapene fra artikkel 4 ble tatt med i spørreundersøkelsen for å undersøke om flere ville si seg enig i disse påstandene (vedlegg C). De tre respondentene er i stor grad enig, men mener at det er viktig å la folk få prøve seg, og at noen av disse egenskapene også bygges opp og modnes gjennom å praktisere PP. Evnen til å kommunisere utpeker seg som den egenskapen alle anser som spesielt viktig. Viktigheten av god kommunikasjon støttes også av Talbott, som mener at en levende dialog er essensiell i PP [6].

Personlige egenskaper er et vanskelig og komplekst område, og vegen er kort mellom fagfeltene informatikk og psykologi når det gjelder PP. For å holde meg til informatikk vil jeg i første rekke støtte momentene om at kommunikasjonsevner er viktig i PP.

Argumentene om at visse egenskaper modnes over tid er også rimelige. I arbeidslivet vil man i mange yrker måtte bruke tid til å tilpasse seg og lære seg å takle egen usikkerhet.

Momentene som angår menneskelige faktorer er basert på erfaringer, og det er ingenting som tilsier at argumentasjonen bør trekkes i tvil. Det som er viktig i forhold til dette punktet er at man er klar over det menneskelige aspektet, og at det finnes utallige varianter av juniorer og seniorer med ulik erfaring og personlige egenskaper.
Det som trekkes frem her må anses som generelle tendenser som kan være rådgivende, men konkrete konklusjoner er verken mulig eller riktig å trekke her.

4.4 Viktige forutsetninger

Under dette punktet diskuteres momenter som i kildene anses som viktige eller kritiske for å lykkes med PP. Graden av viktighet vil være avhengig av mange ulike faktorer, og det som blir ansett som kritisk i en sammenheng kan klassifiseres som mindre viktig i en annen.

4.4.1 Rullering og rollebytte

Rullering av parsammensetning og rollebytte internt i paret praktiseres på flere måter. I hovedsak er det enten styrt av en leder eller coach (XP). Rullering og rollebytte kan både være situasjonsbetinget og oppgavebetinget.

Uansett hvordan man velger å gjøre dette, så poengteres det i flere av kildene at dette er en viktig forutsetning for å ivareta kunnskapsoverføringen i teamet [7][11](vedlegg C). Det påpekes også at en konsekvens av at man ikke rullerer i paret kan bli at navigatøren blir sittende passiv, noe som også kan gå ut over kodegjennomgangen.

4.4.2 Samarbeidsgrad

Flere av kildene omtaler grad av samarbeid. Problematikken her er knyttet til om all kode skal utvikles med PP, eller om man bør bruke IP på enkelte oppgaver.

I artikkel 1 [4] refereres synspunkter fra programmerere om bruk av IP. I mange tilfeller må man bruke IP på grunn av sykdom, tidspress og lignende. Det fremheves her at det er viktig å prioritere hvilke deler av utviklingen man skal bruke PP på, og at man på forhånd er enig om hvordan man skal håndtere det som eventuelt må gjøres med IP. Analyse og design utført med PP hevdes å være en kritisk suksessfaktor (ibid). Dette momentet er troverdig og underbygges også av funnene i artikkel 3 [9].

Artikkel 2 [8] er preget av en del usikre momenter, de presenterer svært høye tall, og oppgavetypen er ikke klart definert. Det ligger imidlertid en indikasjon her på at trivielle oppgaver kan løses mer effektivt med IP.

Artikkel 3 [9] omhandler studier direkte rettet mot kompleksitet i oppgaven, og funnene her viser at jo mer komplekse og tankemessig utfordrende oppgaver - jo mer effektivt vil det være å bruke PP. På enkle oppgaver som krever lite analyse og tenking hevdes det her at PP er ineffektivt, og argumentasjonen for dette er solid.

Erfaringene fra Proxycom i artikkel 8 [12] er at oppgavens kompleksitet er avgjørende for om PP er effektivt, og de anbefaler derfor å bruke IP på enkle oppgaver. Selv om disse erfaringene er fra kun ett prosjekt, vil de kunne regnes som en understøttelse for andre funn.

Artikkel 9 (vedlegg C) anbefaler PP til alle oppgaver i utgangspunktet, fordi det som blir produsert med IP ofte må gjennomgås etterpå uansett. Her foreslås det også at all enkel kode bør genereres automatisk, resten kan skrives i par fordi det skal lite til før en støter på kompleksitet, selv i små oppgaver. Disse synspunktene kan ha sammenheng med at PP brukes som en del av XP, og det i XP er et relativt strengt krav om å skrive all kode i par.

Dersom man velger å innføre PP, bør man ta stilling til om man skal bruke innslag av IP, og i tilfelle i hvilken grad. Det er stor sannsynlighet for at spørsmålet vil komme opp selv om man i utgangspunktet går inn for å bruke PP på all kode (sykefravær etc).
Momenter man bør ta med i denne vurderingen er:

- Hvilke aktiviteter skal prioriteres for PP
- Hvordan klassifisere oppgaver ut fra kompleksitet
- Hvordan skal man etterbehandle kode som er produsert med IP
- hvordan vil bruk av IP påvirke andre aspekter ved PP (for eksempel kunnskapsdeling)

Disse momentene sett under ett vil ikke gi grunnlag nok for å anbefale rendyrket PP. Som et meningsinnlegg i forhold til dette punktet viser jeg også til Beck [1]. Hans erfaring er at PP er mer produktivt enn om man jobber individuelt med deler av koden og deretter integrerer resultatet. For å imøtekomme skepsisen som ofte er rettet mot PP sier Beck at først må PP læres skikkelig, så kan man prøve å bruke PP på all produksjonskode i en iterasjon, deretter IP på en annen iterasjon. Da først har man et utgangspunkt for å gjøre seg opp en mening (ibid).

Det at PP må læres før man kan forvente full effekt av fordelene, anser jeg som et viktig poeng.

4.4.3 Prosjektstørrelse

I den informasjonen jeg har samlet inn, er det svært lite å finne om PP i forhold til prosjektstørrelse. Med prosjektstørrelse menes det her antall medlemmer i PP-teamet.

Williams et. al [4] mener at store prosjektgrupper skulle kunne dra nytte av PP, men at det er nødvendig med forskning på dette området.
Johansen [13] foreslår en begrensning i prosjektstørrelsen på maksimalt ti personer i sin håndbok for XP. Dette begrunner han med at det kan bli for mange som skal ha oversikt over koden, og momentet er dermed direkte knyttet til PP.

Prosjektstørrelse nedad er heller ikke mye omtalt, men mye av argumentasjonen for fordelene med PP bygger på at det er flere enn to utviklere i teamet (som er det innlysende minimum). Det er rimelig å anta at fordelene ved for eksempel kunnskapsspredning, lavere risiko ved turnover, kollektivt eierskap og integrert opplæring vil være proporsjonale til antall utviklere i teamet.

I artikkel 6 vises det til at jo flere medlemmer det er i PP-teamet, jo mindre tid må man tilbringe i par med dårlig personkjemi, selv om det brukes styrt rullering [11]. Beck [1] mener at dersom to personer ikke kommer overens, så skal de ikke jobbe sammen i par.

Dette tilsier at i et lite team vil man ha større vanskeligheter med å håndtere personkonflikter dersom det skulle oppstå.

Disse momentene gir visse indikasjoner, men manglende forskning og dokumentasjon gjør at det ikke kan trekkes noen konklusjon i forhold til prosjektstørrelse.

4.4.4 Utviklingsmetode

I denne oppgaven har ikke ulike metodeverk rundt PP vært et fordypningstema, men det er likevel verdt å merke seg enkelte innspill fra erfaringsrapportene. Det ser ut til at PP er mest brukt i sammenheng med XP, noe som anses rimelig ettersom PP er en integrert teknikk i denne metoden.

Blant de spurte programmererne er meningene delte om PP i forhold til metode. Respondent C mener at PP kan brukes i enhver metode, mens respondent A mener at en variant av XP vil passe best etter at han har prøvd PP i ulike kontekster. Han vektlegger her behovet for en metode som tilrettelegger for den nære koblingen mellom utvikling av design og implementasjon.

Spørsmål om metode ble også stilt til en av forfatterne av artikkel 4 [7] og selv om han anbefaler XP, mener han at PP også kan brukes som frittstående teknikk. Han poengterer at det da er viktig å bruke noen ”best-practices” i tillegg for å unngå problemer: kodestandard, felles forståelse av domenet (metafor) og enkelt design (vedlegg A).

Om variabelen Systemutviklingsprosess:
PP vil i de fleste tilfeller være en del av en systemutviklingsprosess, for eksempel XP eller RUP. Den prosessen PP inngår i vil kunne sette begrensninger og/ eller bruke andre aktiviteter som kan påvirke PP-aktiviteten. Dette kan igjen påvirke resultatene PP, og den prosessen man bruker kan derfor vise seg å være svært viktig [14].

Ut fra disse momentene går en klar anbefaling om å være bevisst på kombinasjonen PP og metode. Bedrifter som vurderer om PP vil kunne lønne seg for dem, bør se dette i sammenheng med både etablert utviklingsmetode og –miljø. Det finnes minst ett rammeverk som foreslår hvordan man kan gjennomføre en test i forhold til dette. Formålet med dette rammeverket er å gi støtte for en vurdering av hvordan innføring av PP kan innvirke på den eksisterende systemutviklingsprosessen i bedriften [15].

4.4.5 Arbeidsmiljø

Rammeverket [14] beskriver det fysiske arbeidsmiljøet som en variabel som kan påvirke effekten av PP. Det vises til en studie hvor 75 % av deltakerne mente det ville være fordelaktig med en ekstra skjerm i PP for lettere å kunne bytte mellom ulike applikasjoner etc. Jeg har ikke funnet noe i mine kilder som støtter dette, men i visse sammenhenger kan dette trolig ha en nytteverdi.

Et annet moment som trekkes frem her er åpent kontorlandskap (ibid). Dette er ofte typisk for miljøer som praktiserer høy grad av samarbeid. En del studier har imidlertid vist at dette kan virke negativt på effektiviteten blant annet på grunn av økt stress og mer støy (ibid).

Arbeidsmiljø er også omtalt i vedlegg C. Det fysiske arbeidsmiljøet bør være prosjektorientert, men dette krever en bevisst holdning i forhold til støynivået. Videre hevdes

det at bedriften må ha en gjennomgående samarbeidskultur for å kunne lykkes med PP (vedlegg C).

I the Software Factory, fra artikkel 6 [11], er fysisk arbeidsmiljø et viktig poeng. Ideen med denne ”fabrikken” bygger på Thomas A. Edisons ”the Invention Factory”, som nettopp vektla et åpent og samarbeidspreget miljø (se www.menloinstitute.com).

Samarbeid er essensen i PP, og det er liten tvil om at både fysisk og psykisk arbeidsmiljø må tilrettelegges for å optimalisere effekten ved PP.

5 Analogier til parprogrammering

Analogier kan ofte være til hjelp for å få økt forståelse for en situasjon. Ved å sammenligne med kjente situasjoner kan man få en følelse av hvordan noe er uten å ha prøvd det selv. Jeg har fått noen innspill på dette fra deltakerne i spørreundersøkelsen, og noe av dette gjengis her.

En klassisk analogi til PP er bilkjøring med en sjåfør og en kartleser. Denne analogien gjenspeiles også i de veletablerte rollenavnene driver og navigatør, og poengterer at den som skriver ”kjører” og er konsentrert om nåsituasjonen. Navigatøren (kartleseren) har et videre perspektiv ved at han tenker litt lenger frem og har oversikt over hvor de er ”på kartet”.

Dette dekker ikke alle aspekter ved PP. For eksempel illustrerer den ikke den viktige dialogen som er i PP og heller ikke at paret treffer beslutninger sammen. I tillegg bør en analogi også betone det aktive samarbeidet i paret. I rallykjøring er for så vidt både sjåfør og kartleser aktive, men i fastsatte roller.

En analogi som passer for å beskrive noe av dette er musikalsk improvisasjon: to musikere som spiller samme stykke på ulike instrumenter. De ”kommuniserer” gjennom musikken, tolker hverandres signaler og skaper noe sammen. De kan også veksle på å lede improvisasjonen videre, eller at man leder hver sine stykker.

Enhver kreativ, komplisert oppgave bør være sammenlignbar med PP. Oppgaver eller situasjoner der man trenger en annen person til å kritisere det man gjør fordi man har for stort eierskap til å kunne være kritisk nok selv. Eller at man sammen med en annen kan diskutere seg fram til en god løsning på oppgaven.

6 Konklusjon

Gjennom denne oppgaven mener jeg å ha vist at parprogrammering kan gi fordeler både for bedriften og dens ansatte.

Bedriften kan oppnå fordeler gjennom en bedre kunnskapsforvaltning, mindre risiko ved turnover, lavere kostnader ved opplæring og styrket konkurranseevne gjennom kortere leveringstid og bedre kvalitet på produktene.

De ansatte – utviklerne – kan oppnå fordeler gjennom bredere kompetanse- og erfaringsområde, og økt trivsel.

Ordet *kan* er brukt helt bevisst her, fordi fordelene med PP kommer ikke av seg selv. Gjennom oppgaven viser jeg også at det er en rekke viktige forutsetninger for å oppnå fordeler ved PP. Forutsetningene er knyttet til kontekst, teknikker, metoder og den menneskelige faktor.

Det er ikke nok å plassere utviklerne i par foran en skjerm og forvente fullt utbytte. En bedrift som vurderer å innføre PP, må la alle involverte få tilstrekkelig kunnskap om PP. Det er viktig å være klar over at PP er en omfattende utviklingsteknikk, og mye mer enn bare programmering. Man må også ha tålmodighet og la utviklerne venne seg til en ny måte å jobbe på.

Siden parprogrammering er en relativt ny teknikk vil det være svært interessant å følge den videre utviklingen. Ikke minst i forhold til hvordan de ulike fordelene vil holde seg over et lengre tidsperspektiv.

”There is always a way to do things better. Find it.”

Thomas Edison

Litteraturliste

1. Beck, Kent (2001) *eXtreme Programming explained: Embrace Change*, Addison-Wesley

2. Jensen, Randall (2003) A Pair Programming Experience (Internett), Crosstalk, Mars, 2003. Tilgjengelig fra: <http://www.stsc.hill.af.mil/crosstalk/2003/03/jensen.html>

3. Williams, Laurie, Robert R. Kessler (2002) Pair programming Illuminated, Addison-Wesley

4. Williams, Laurie, Robert R. Kessler, Ward Cunningham og Ron Jeffries (2000) Strengthening the case of Pair Programming (internett) Tilgjengelig på: <http://collaboration.csc.ncsu.edu/laurie/Papers/ieeeSoftware.PDF>[lest 4.3.2204]

5. Nosek, John T. (1998) The Case for Collaborative Programming (Internett) Tilgjengelig fra <http://proquest.umi.com via http://www.hihm.no> [lest 13.4.2004]

6. Talbot, Nathaniel (2001) Conversant Pairing (Internett) Tilgjengelig fra: <http://pairprogramming.com/conversantpairing.htm>[lest 13.4.2004]

7. Dick, Andrew J., Bryan Zarnett (2002) Pair Programming & Personality Traits (Internett) I Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in SoftWare Engineering, side 82-85. XP2002 Tilgjengelig fra: <www.redhookgroup.com/resources> [lest 25.3.2004]

8. Smith, Randy, Allen Parrish, Joanne Hale og David Hale (2003) Pair Programming: Pairing alone is not enough. IEEE Software 13. august 2003. Tilgjengelig fra: <http://cs.ua.edu/9TechnicalReports.shtm >[Lest 1.4.2004]

9. Lui, Kim Man and Keith C.C. Chan (2003), When Does a Pair Outperform Two Individuals? (Internett) Proceedings of XP 2003 (Springer LNCS 2675), side 225 – 233 Tilgjengelig fra: <http://www.soe.ucsc.edu/~brianh/PairProgramBib.html>

10. Flies, Daniel D (2003) Is Pair Programming a valuable Practice? (Internett) Tilgjengelig fra: <http://csci.mrs.umn.edu/UMMCSciWiki/pub/CSci3903s03/StudentPaperMaterials/flies-pairprogramming03.pdf > [lest 25.3.2004]

11. Meloche, Thomas, James Goebel and Richard Sheridan (2003) Paired programming in the Software Factory (Internett) Tilgjengelig fra <http://www.menloninstitute.com/freestuff/whitepapers/> [lest 30.3.2004]

12. Johansen, Trond (2002) Ekstrem Programmering – Praktiske erfaringer.

13. Johansen, Trond (2002) Håndbok i Ekstrem programmering.

14. Gallis, Hans, Erik Arisholm og Tore Dybå (2003) An Initial Framework for Research on Pair Programming. Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE'03)

15. Succi, Giancarlo, Milorad Stafanovic, Michael Smith, Richard Huntrods (2001) Design of an Experiment for Quantitative Assessment of Pair Programming Practices. (Internett) Publikasjon på XP2001. Tilgjenglig fra: <http://www.agilealliance.com/articles/articles/QuantitativeAssessment.pdf>

16. Nawrocki, Jerzy og AdamWojciechowski (2001) Experimental evaluation of pair programming. (Internett) Tilgjengelig fra: <http://www.agilealliance.com/articles/articles/ExperimentalEvaluationOfPP.pdf >

17. Williams, Laurie, Alistair Cockburn (2000) The Costs and Benefits of Pair Programming (Internettt) Tilgjengelig fra: <http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF>[lest 13.3.2004]

18. Tønnesen, Eva (2004) Tema: Kunnskapsdeling Når følelser slår ut på bunnlinja. *It-kurs* nr 2, 2004.

Mail til Andrew Dick hos Red Hook Group, med hans svar innlagt i opprinnelig tekst (i kursiv)

Hi Anne,
I am glad that you found it interesting and you may certainly reference it for your paper. Here is the publication information from XP2002:

Andrew Dick and Bryan Zarnett (2002).
Paired Programming and Personality Traits.
In Proceedings of the 3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering, pages 82-85.

I'll intersperse my remaining comments below...

> *Hello!*
>
> *I'm a computer science student from Norway and I am writing a paper based on empirical reports, articles and studies. My special topic is pair programming (PP) and I'm concentrating on which benefits an organization can achieve by using PP. I would also like to find out if there are any critical factors for success. It isn't very easy to find anything about success criteria but I have just read your article "Paired Programming & Personality Traits" which I found at www.redhookgroup.com. I found it very interesting and I am wondering if it's allright for you if I use it as a reference?*
>
> *I have a few questions I would like to ask you and I hope you find time to send me an answer.*
>
> *I'm writing about PP in general – not particularly as a part of XP as you do in the article. I would think that those personality traits pointed out are important to PP in general too – do you agree?*

I think that the personality traits are quite important to PP in general as you say. I was doing PP before Kent Beck's first XP book came out, but didn't know that was what it was called. A few developers and myself had fallen into the habit of working in pairs and were accomplishing a great deal. Beck's book gave a term for what we were doing and explained a bit more why it was beneficial. After attempting a full XP project however we found that PP isn't easy and that numerous pairs were failing to achieve the same performance level. We identified those traits as some base qualities that were present in the pairs that did well, and lacking in the pairs that did not.

> *And would you say that having developers with these traits is critical for the success of PP?*
> *(You almost say it in the article but in another words…)*

I would say it is critical but I wouldn't say it is essential. I am sure there are at least a few developers who don't have those traits who would do fine PPing - there are always exceptions. :)

> *If you should give me another (one or two) critical success factors, what would it be ?*

Probably that extroverts would likely do better PPing (although that may be because they tend to possess a number of the discussed personality traits). In addition, I have found that a desire to learn / improve helps. I find that I can PP better with developers that treat a PP session as an opportunity to learn new ways of doing things and to refine their skills. I find I can still learn the occasional thing even from a junior programmer (even it is just explaining why you are solving a problem in a particular fashion - their questions may spark improvements).

> *In the summary of the article I read: "XP is comprised of best practices that work well together but are ineffective on their own." Does this mean that you wouldn't recommend using PP without using XP?*

I find most of the XP practices balance each other and that using one (to the extent required in XP) alone can introduce new problems or consequences. That being said, some practices can work well in an agile development environment without significant drawbacks - unit testing & test driven development, coding standards, continuous integration. I wouldn't say you couldn't do PP without using XP (after all I started out that way), but you would need some best practices in place to reduce conflict (coding standards, common understanding of the domain (metaphor), simple design).

Another factor with PP that I have found is that you cannot force it on a group of developers (I have experienced watching management attempt to do this). Yet that same group of developers that would insist they couldn't benefit from PP, probably does PP a good portion of their day without realizing it - whenever a tough bug / design problem comes up, they will often call a neighbouring developer over to help out.

> *I'd be very thankful if you would send me a reply on this mail.*

Good luck on the paper.

Andrew

> *Best regards*
> *Anne Kongsrud*

Mail til Thomas Meloche hos Menlo Institute, etterfulgt av hans svar

Hi
I am a Norwegian computer science student. In this semester we are writing about one special topic within software development and I'm writing about pair programming (PP). We are supposed to evaluate research reports, articles etc based on experiences from "real life". At the Menlo Institute's homepage have I found the whitepaper "Paired Programming in the software factory" which is of great interest for my paper. I have a few questions about success factors and would be glad if you would send me a reply.

The whitepaper emphasise the forming of pairs. My opinion after reading your paper is that right pairing is one critical factor of success for PP. Do you agree?

Are there any other critical factors you can think of?

Thank you!

Anne Kongsrud

Svar fra Meloche:

Right pairing is critical for the long term life of the project, i.e. to spread information around the team, survive changes, and improve the code.

However, on any given iteration it is not necessary to worry about the details of the pairs as a critical success factor, i.e. do not worry if you have the right two people assigned to the right task.

Most of the other XP practices are, in fact, critical factory.

1. Co-location
2. Code Stewardship
3. Do the simple thing -You aren't going to need it
4. Changing pairs
5. Changing assignments

etc.

We've found PP works best taken as one of the many practices within XP.

Best of luck with your paper. Please send a copy when you are done, if you'd like, I'll post it on the website.

-Tom

Thomas G. Meloche
President, The Menlo Institute LLC
www.menloinstitute.com

Vedlegg B

Forespørsel som ble sendt angående spørreundersøkelse.

Adresseliste: Speakanet (danmark): info@speakanet.com
Callista It-partner: itpartner@callista.se
Peter Lindberg: peter@tesugen.com
Antares(Norsk): info@antares.no

Hei!
Jeg er student ved Høgskolen i Hedmark (Norge) hvor jeg studerer informatikk med fordypning i Systemutvikling. Dette semesteret jobber vi med kunnskapsbasert systemutvikling, og vi skal i den forbindelse skrive en oppgave i et selvvalgt spesialemne.

Vår foreleser er professor Magne Jørgensen som arbeider ved Simula Research Laboratory i Oslo. Jeg har valgt temaet Parprogrammering og min oppgave kommer til å dreie seg om hvilke fordeler parprogrammering kan gi fremfor individuell programmering. Jeg vil også forsøke å beskrive kritiske suksessfaktorer i forhold til parprogrammering. (Parsammensetning, fysisk arbeidsmiljø etc) I hovedsak vil jeg benytte meg av tilgjengelig litteratur i form av bøker, empiriske artikler, forskningsrapporter osv.

I tillegg er jeg svært interessert i å komme i kontakt med bedrifter/ personer som har erfaring med parprogrammering, slik at jeg også kan bygge min oppgave på slike referanser.
[firmanavn] har jeg funnet gjennom søk på temaet parprogrammering på Internett. Mitt spørsmål til [firmanavn/ deg]er om [noen hos dere/du] har mulighet til å være et referansepunkt for meg? Jeg vil anta at dette ikke vil bli veldig tidkrevende, da det trolig vil dreie seg om å svare på noen få (5-10) spørsmål, eventuelt innspill/kommentarer til korte, konkrete problemstillinger.

Oppgaven skal leveres 4. mai, slik at perioden 25. mars – 25. april vil være mest aktuell for informasjonssøk og kontakt med referanser, med hovedtyngde i første del av perioden.
Jeg vil være svært takknemlig for alle mulige innspill – og jeg vil også sette pris på om jeg får svar på denne mailen uansett om du/dere kan hjelpe meg eller ei.

Med vennlig hilsen
Anne Kongsrud

Alle bortsett fra Callista It-partner svarte positivt på henvendelsen.

Forespørsel som ble sendt Proxycom

proxycom@proxycom.no

Hei
Jeg er student ved Høgskolen i Hedmark og skal skrive en oppgave om parprogrammering basert på empiri. På Proxycoms websider ligger denne linken: "Evaluering av metodikken Ekstrem programmering". Linken er "død", men jeg lurer på om det likevel er mulig å få denne informasjonen?

Med vennlig hilsen
Anne Kongsrud

Som respons på denne henvendelsen fikk jeg tilsendt en erfaringsrapport og en håndbok for XP utarbeidet av Trond Johansen i Proxycom

Dette dokumentet inneholder spørsmålene som ble stilt til programmererne. Overordnede spørsmål om erfaring ble stilt i forkant og dette er fremstilt i tabellen nedenfor.

Respondentene er:
Peter Lindberg (A), Morten Ulrik Sørensen, Speakanet (B) og Stig Tollefsen, Antares (C).

Respondent	Programmeringserfaring			Rolle
	Totalt	Profesjonelt	Herav PP	
A	17 år	10 år	3-4 år	Utviklingssjef
B	13 år	10 år	3 år	Coach/ prosjektleder/utviklingssjef
C	23 år	8,5 år	ca 1 år	Utvikler

Svarene fra hver enkelt er gjengitt under hvert spørsmål. De tre programmererne er fra Sverige, Danmark og Norge og jeg har valgt å gjengi svarene på originalspråket, fortløpende for hvert spørsmål.

Spørreskjema

Fordeler med parprogrammering (PP)
Av det jeg hittil har lest om PP er disse de oftest omtalte fordelene:

høyere kvalitet på koden (høy lesbarhet og enklere vedlikehold)
gladere og mer fornøyde programmerere
spredt kunnskap (ingen har spisskompetanse = lavere risiko hvis noen slutter etc)
kortere utviklingstid
færre feil i sluttproduktet

Er dette fordeler du selv har opplevd?

A: Det blir absolut kortare utvecklingstid, dels eftersom man arbetar ”intensivare” i par, dels eftersom man kan (förutsatt att utvecklingsmetoden och projektet tillåter det) lämna mer av designen öppen tills dess att man implementerar, dvs. fatta designbeslut under implementeringsarbetet.
Gladare och mer nöjda utvecklare tycker jag hänger på att kemin mellan utvecklarna fungerar. Parprogrammering ställer väldigt höga krav på den sociala interaktionen. Det krävs att båda utvecklarna, även ”bisittaren”, spelar en aktiv roll i arbetet. Jag har parprogrammerat med personer som varit väldigt passiva och då jag fått se till att engagera dem med frågor om synpunkter, osv., vilket gör att arbetet känns trögare. Men jag vill poängtera att jag parprogrammerat med programmerare som jag uppfattat som ”ensamvargar”, dvs. att jag trott att de skulle föredragit att arbeta individuellt, men där det har blivit ett väldigt stimulerande samarbete. Men rent generellt tycker jag att det stämmer att stämningen blir bättre; i utvecklingsprojekt kan det ibland bli så att man endast samtalar med varandra på formella möten och på kaffe- och lunchraster.
Spridd kunskap kräver att man ”roterar” paren, dvs. att man varierar parkonstellationerna så att ”kunskapsöverföring” kan ske. I de projekt jag har deltagit i har det varit en ganska fast rollfördelning, trots att man velat tillämpa XP, dvs. att specifika personer haft ansvar för särskilda områden, och då har parkonstellationerna tenderat att bli fasta och därmed att kunskap inte har spridits så effektivt som det skulle kunna i teorin. Jag tror det krävs att man tillämpar XP eller någon annan ”agile”-metod för att denna aspekt av parprogrammeringen ska kunna realiseras.
Färre fel och högre kvalitet som resultat av parprogrammering per se har jag svårt att uttala mig om eftersom vi alltid har tillämpat ”testdriven utveckling” (”test driven development”) vilket reducerar antalet buggar markant och leder till högre kvalitet. Men jag kan tänka mig att parprogrammering bidrar till det i viss utsträckning, men kanske mer i form av att designen blir bättre anpassad till verkligheten när man tillåter sig att fatta en större del av designbesluten under implementeringsarbetet (vilket återigen är avhängig utvecklingsprocessen).

B: Ja, alle sammen.

C: Jeg har opplevd punkt 1 og 3; de andre punktene krever mer trening enn vi hadde før prosjektet reverterte til enkeltprogrammering. Fornøyde programmerere får man først når teamet er innkjørt, og det krever også at de som parres er på likt nivå - hos oss ble det forventet at ene halvparten skulle gjøre opplæring samtidig, og da fungerer det ikke. Kortere utviklingstid tar også noe innkjøringstid å oppnå; vi så at hastigheten steg etter hvert. Færre feil har jeg ikke grunnlag for å kunne konkludere med, men se under.

Har du opplevd andre fordeler med PP?

A: Jag tycker att punkten med ”gladare och mer nöjda utvecklare” dessutom innehåller dimensionen att det blir bättre sammanhållning i teamet, att man odlar en kultur där man hjälps åt och projektets välgång är något som angår alla, till skillnad från att var och en månar om sina delar av det hela systemet. Skillnaden mellan ett framgångsrikt projekt och ett misslyckat är ofta att man lyckas med att förena teamet och få samtliga att arbeta för att lyckas leverera helheten och inte bara delarna. Det är en viktig skillnad.

B: At vi hjælper hinanden til at holde fast i den høje kvalitet, og holde fast i at gøre tingene på den rigtige måde.

C: Den største fordelen er at man faktisk får gjort kodegjennomgang. Uten parprogrammering blir dette i beste fall gjort sporadisk - og den som skal gjøre kodegjennomgangen går fort lei og begynner å slurve. Det er også mye enklere å oppdage feil når man er der i prosessen, og man slipper å bruke masse tid på å forstå hvordan koden skal virke (uansett hvor godt dokumentert ting er, kan dette være vanskelig). Det er den kompliserte koden som mest trenger kodegjennomgang, og det er den som er vanskligst å finne feil i ved kodegjennomgangen.

Kritiske suksessfaktorer

Fordelene med PP kommer vel ikke av seg selv. Det jeg hittil har funnet om kritiske suksessfaktorer er det som går på menneskelige faktorer:

Personlige egenskaper

Andrew J. Dick har skrevet en artikkel hvor han trekker frem fire egenskaper som han mener er kritiske suksessfaktorer for PP:
- god kommunikasjonsevne
- må være komfortabel med å jobbe sammen med andre
- må være trygg/ sikker på egne evner (dvs: ikke være redd for at egen kode ikke er god nok)
- må ha evne (og vilje) til å kompromisse (spesielt når det gjelder å forkaste egen kode)

Er du enig i dette? Er det eventuelt andre egenskaper/ menneskelige forhold du anser som like viktige/ viktigere?

A: Ja, jag är enig, men vill poängtera att man inte ska dra förhastade slutsatser om specifika personers lämplighet för parprogrammering. Det hänger väldigt mycket på att man provar på. Ofta förstår man inte hur det är förrän man har provat. Jag känner till flera som själva blivit överraskade av att de tycker om detta sätt att jobba, som trott att de föredrar att arbeta solo. Och att vara trygg/säker på sin egen förmåga är något som om inte annat kommer med tiden. Man får inte glömma att parprogrammering är ett utmärkt sätt att lära sig programmering: man har en ständigt närvarande mentor.
Vilja att kompromissa skulle jag vilja säga handlar om en öppenhet gentemot andras förslag, och gentemot att prova idéer som kanske verkar orimliga, osv.

B: Det er vigtigt, men også i noget omfang selvopfyldende: Parprogrammering med en god partner kan hjælpe en til at blive bedre til at kommunikere, at få det bedre med at arbejde sammen med andre, blive sikrere på egne evner, og lære en at gå på kompromis.
Det er dog selvfølgeligt langt lettere hvis alle har de nævnte egenskaber i forvejen. Og meget svært hvis man har problermer på flere af de 4 områder.

C: Det er klart at det hjelper å kunne samarbeide, men dette gjelder uansett i et prosjekt med flere mennesker. Jeg vil si at det viktigste er at de to som skal jobbe sammen er på samme nivå teknologisk. Greier man ikke å

kommunisere, vil mye annet gå galt også. Forkaste egen kode - dette er en evne som ofte trenges også ved enkelprogrammering.

Rullering av par og roller

(Rullering = bytte av partner/ rolle).
Bytte av partner er åpenbart en viktig forutsetning for å oppnå fordelen kunnskapsoverføring. Bytte av partner kan for eksempel skje:
- for hver oppgave
- for hver iterasjon
- for hver dag
- frivillig
- styrt (av coach/ eller følger en bestemt rullering)

Hva er din erfaring? Er det noe som er bedre enn noe annet her – og i tilfelle hvorfor?

A: Vilka par man arbetar i tycker jag i första hand ska vara frivilligt. Personligen skulle jag inte vilja att någon bestämde vem jag skulle arbeta tillsammans med. Men jag inser att det kan leda till att parkonstellationerna blir statiska och då är det nog bra om coach eller projektledare kan poängtera att det är dags att rotera paren lite, men då låta det vara upp till alla att hitta partner.

B: Vi skifter frivilligt, men er dog stærkt påvirket af at vi tit kun er to personer i et par iterationer i træk – så bytter vi naturligvis ikke så meget...
Det eneste tilfælde hvor vi har styret partner- og rollebytte har været når der har været en på et projekt som f.eks. ikke var helt faldet til (endnu), eller for hvem parprogrammering ikke er helt naturligt endnu (se sidste spørgsmål) – så har vi tænkt over hvem det ville være godt at sætte vedkommende sammen med. Ellers kører det ”af sig selv”.

C: Da vi gjennomførte dette, rullerte vi forholdsvis mye. Likevel viste det seg at de tekniske kunnskapene var så vidt spredt at mange oppgaver gav seg selv. Det er vanskelig å peke på kunnskapsoverføring i en budsjettsammenheng (at det koster noe), så der er definitivt et press til å la være å rullere for mye - selv om det i lengden er godt for prosjektet. Hos oss styrte jeg rulleringen (jeg var coach), og vi byttet hver iterasjon. Vi hadde korte iterasjoner, så en oppgave tok vanligvis en iterasjon å gjennomføre.

Rollebytte *(driver/ navigatør) synes i mange tilfeller å være styrt av erfaring. Noen mener at den minst erfarne i paret bør være driver, mens andre mener det motsatte. Oppgavens karakter kan også være avgjørende. Det finnes også de som ikke bruker faste roller i det hele tatt, men sender mus og tastatur frem og tilbake ettersom det passer.*

Hva mener du? Er det viktig at hver partner har sin bestemte rolle, og i hvilken grad bør kunnskap/ erfaring legges til grunn?

A: En av fördelarna med parprogrammering tycker jag är att man kan växla roll flera gånger under en dag. Det är nästan så att det är en förutsättning för att man ska klara av intensiteten, att man kan säga att man vill vara bisittare ett tag och låta den andre ”köra”. Jag kan inte se någon särskild poäng med att låta den mest eller minst erfarne alltid köra. Däremot är det viktigt att uppmuntra den minst erfarne att uttrycka sina synpunkter, utan rädsla för eller ödmjukhet gentemot mer erfarna. Jag har varit med om många tillfällen då riktigt bra idéer har kommit från oerfarna utvecklare som vågat dela med sig av dem.

B: Vi kører frem og tilbage som det passer. Med en erfaren og en uerfaren og et tidspres kan det dog kræve en bevidst beslutning at sørge for at lade den mindst erfarne komme til.

C: Det var her jeg så de negative sidene (eller hva som ikke virket). Hvis paret ikke er godt balansert, vil den med minst erfaring ofte ikke greie å følge med. Da bør han eller hun ta styringen. Men vi fikk ikke gjort dette lenge nok til at vi fikk rutine på det. Er spranget for stort, kan det bli bra opplæring av det, men ikke særlig effektiv utvikling. I balanserte par så jeg det som fungerte best - da ble tastatur gjerne sendt frem og tilbake i løpet av oppgaven.

Hva annet vil du eventuelt trekke frem som kritiske suksessfaktorer for PP? (Prosjektstørrelse, fysisk arbeidsmiljø, størrelse på team, utviklingsmetode etc)

A: Det hänger väldigt mycket på företagskulturen om det ska lyckas väl. Vissa företag har en kultur där man arbetar individuellt med ganska låg grad av samarbete. Att då förespråka parprogrammering kan bli svårt. Jag har varit med om att man haft för avsikt att tillämpa parprogrammering, men där högre direktiv krävt att specifika personer ska flyttas till andra projekt, osv.
Att tillämpa parprogrammering leder till ökad grad av kommunikation mellan teammedlemmar, vilket möjliggör effektivare möten, osv., men detta drar inte alla nytta av. Jag tycker nog att parprogrammering passar bäst i en utvecklingsmetod à la XP, där man har för avsikt att göra mer av designarbetet i samband med implementeringsarbetet, att tillämpa effektivare former av projektplanering, möten, osv.

B: Gensidig tillid mellem alle team-medlemmer skal enten være der eller skaffes – det er kritisk.

C: Jeg tror det er viktig at prosjektet forstår at parprogrammering er i stedet for kodegjennomgang. Det som også blir kritisk, er prosjektets syn på eierskap av kode. For sterk individuelt eierskap vil ødelegge mye av effekten av parprogrammering. Vi prøvde å rotere nok til at prosjektet fikk et kollektivt eierskapsforhold til hele kodebasen. Det fysiske arbeidsmiljøet bør også være prosjektorientert (versus faste kontor), slik at det blir enkelt å bytte arbeidsplass. Personlig synes jeg PP bør kunne brukes med fordel i enhver utviklingsmetode. Ellers er det jo viktig at folk lærer seg å ikke bli for høyrøstet.

Individuelt arbeid

Det er ulik praksis for om man til enhver tid jobber i par. Noen mener det er mer effektivt å løse enkle oppgaver individuelt. En måte å gjøre dette på er at paret selv sorterer oppgavene og bestemmer hva de kan gjøre individuelt.

Mener du at enkel kode bør skrives individuelt? Hvilke fordeler og ulemper ser du?

A: Det är klart att enstaka uppgifter går smidigare att göra individuellt, men överlag så föredrar jag parprogrammering fullt ut. Det är en väldigt liten procent av den totala tiden som är icke-parprogrammering.
Jag har varit med om fall då vi tillämpat parprogrammering, men där den ene partnern jobbat över och dagen efter gjort en hel del saker på egen hand, som då behövts gå igenom och diskuteras, vilket gjort att det inte varit någon fråga om större tidsbesparing.
Jag tycker det är bra om att man enas om en nivå för parprogrammering och att alla håller sig till den. Det är ju viktigt att alla kommer och går ungefär samma tider, osv.

B: Vi foretrækker at al kode så vidt som muligt skrives i par, men er tit et ulige antal – og i så fald forsøger vi at det fortrinsvis er enkle opgaver som løses af en enkeltperson. Desuden prøver vi at sørge for at man ikke er alene alt for længe, så når vi er et ulige antal er der gerne en der skifter mellem to par.

C: Jeg mener enkel kode bør genereres og ikke skrives. Det skal likevel ikke mye til før koden blir komplisert nok til at det kan oppstå feil, og da hjelper PP. Eksempler på kode som kan genereres er stubs (local/remote interfaces) og enkle value objects. Føler man at man kan skrive koden uten å tenke, er det på tide å finne en måte å generere den på, for det er kjedelig arbeid.

Utviklingsmetode

Er erfaringen du har med parprogrammering knyttet til en bestemt utviklingsmetode – og i tilfelle hvilken?

A: Ja, med Extreme Programming (XP), även om jag har tillämpat parprogrammering i projekt som långt ifrån har varit fullskaliga tillämpningar av XP.

B: Ja, Extreme Programming.

C: Den er knyttet til eXtreme Programming, men vi var ikke i stand til å benytte alle elementene i XP.