Utvärdering av parprogrammering

IV1303 Modern mjukvaruutveckling

Larisa Cof
KTH, CINTE

larisac@kth.se

Sara Moazez Gharebagh
KTH, CINTE

saramg@kth.se

Sammanfattning

Parprogrammering är en av de mest omtalade arbetssätten i Extreme Programming (XP) och innebär att två stycken programmerare löser en uppgift tillsammans. Studier har visat att parprogrammering bidrar positivt till mjukvaruprojekt men vi vet väldigt lite om hur mycket parprogrammering påverkar utvecklingshastigheten. I den här rapporten utvärderar vi därför parprogrammering i syfte att undersöka huruvida det är en effektivitetsmässigt gynnsam metod samt för att identifiera dess svaga punkter och föreslå förbättringar. För detta ändamål använder vi oss av de fyra kriterierna: effektivitet, kodkvalité, fördelning av kunskap och motivation och tillfredsställelse. Dessa kriterier är användbara för att ge oss en överblick om vilka brister parprogrammering har. Våra resultat visar att parprogrammering bidrar till en förbättring av kodkvalitén och till mer motiverade och tillfredsställda programmerare om navigatören och föraren ligger på ungefär samma kompetensnivå. Resultaten visar även att parprogrammering bidrar till ett vettigt utbyte av kunskap men är effektivitetsmässigt meningslöst vid enklare uppgifter. Av denna anledning föreslås förbättringar att göra goda tidsuppfattningar inför ett projekt för att därefter kunna identifiera vilka delar av projektet som drar störst nytta av att implementera parprogrammering. En annan förbättring som föreslås är att man under projektets gång skaffar en uppfattning om vilka utvecklare som fungerar bra ihop och baserat på detta föreslår lämpliga par.

Nyckelord: Utvecklingsmetod, Extreme Programming, förare, navigatör, kodkvalité.

Innehållsförteckning

1. Inledning

Mjukvara har varit en viktig del av samhället i flera år. I början utfördes mjukvaruutvecklingsprojekt utan någon riktig planering och utformandet av system bestämdes med många kortsiktiga beslut. När systemen blev större och mer komplicerade blev det svårare att skapa stabila och högkvalitativa produkter vilket medförde att nya sätt att utveckla på introducerades. Idag är de mest använda metoderna inom mjukvaruutveckling de traditionella och agila utvecklingsmetoderna.

De traditionella utvecklingsmetoderna består av ett antal sekventiella steg där kraven definieras i början av projektet. Vissa fann denna metod frustrerande då många projekt kräver frekventa kravförändringar vilket resulterade i att metoder baserade på iterativ utveckling introducerades år 1975. Dessa kallas agila utvecklingsmetoder och möjliggör det att snabbt anpassa sig till förändringar i kravspecifikationen. (Awad, 2005)

Ett arbetssätt som sedan 1995 blivit alltmer populär är parprogrammering som går ut på att två programmerare löser en uppgift tillsammans. Parprogrammering ingår i Extreme Programming (XP) och är en agil utvecklingsmetod som hanterar förändringar genom att leverera mjukvaran tidigt samt genom att utnyttja feedback i utvecklingen. (Williams, Kessler, Cunningham, & Jeffries 2000)

Trots att tidigare studier visat att parprogrammering bidrar positivt gällande kodkvalité och fördelning av kunskap i gruppen är det många företag som inte använder sig av det. Man anser att det inte är ekonomiskt försvarbart att sätta två programmerare på en uppgift som en arbetare kan utföra på egen hand. (Cockburn & Williams, 2001) Problemet är att vi inte vet hur mycket parprogrammering påverkar utvecklingshastigheten och därmed inte hur effektiv och gynnsam metoden egentligen är i ett mjukvaruprojekt.

I den här rapporten undersöker vi, med utgångspunkt från det projekt vi utfört i samband med kursen II1305 på KTH, om parprogrammering är en fördelaktig utvecklingsmetod inom mjukvaruprojekt. Syftet är att utvärdera metoden, jämföra våra resultat med redan existerande studier och föreslå förbättringar inför framtida projekt. Rapporten följer

nedanstående struktur. Kapitel 2 redovisar en teoretisk bakgrund om parprogrammering och dess förväntade nackdelar och fördelar samt en beskrivning av vårt projekt. Kapitel 3 redogör för vår forskningsmetod. Kapitel 4 redovisar våra resultat och kapitel 5 analyserar dem. Slutligen i kapitel 6 presenteras slutord och förslag på förbättringar inför framtida projekt.

2. Bakgrund

I detta kapitel redogörs det för vad parprogrammering är, hur det går till och vilka effekter som påvisats till följd av att ha arbetat enligt metoden. Inledningsvis beskrivs traditionella mjukvaruutvecklingsmetoder (kapitel 2.1) och agila mjukvaruutvecklingsmetoder (kapitel 2.2). Därefter redogörs det för skillnaderna mellan dessa metoder i kapitel 2.3. Parprogrammering beskrivs i kapitel 2.4. I kapitel 2.5 respektive 2.6 beskrivs förväntade för- och nackdelar av parprogrammering. Bakgrunden avslutas med en projektbeskrivning i kapitel 2.7.

2.1 Traditionella mjukvaruutvecklingsmetoder

Traditionella mjukvaruutvecklingsmetoder är baserade på samtliga faser i den så kallade mjukvaruutvecklingslivscykeln (se figur 1). Utvecklingens flöde i denna metod är enkelriktat och följer livscykeln ganska precist. Mjukvaruutvecklingslivscykeln inleds med en planeringsfas där man sedan ställer upp en kravspecifikation. Därefter går utvecklingen vidare mot designfasen. Efter implementationsfasen övergår utvecklingen till testfasen och slutligen underhållsfasen.

Figur 1. Mjukvaruutvecklingslivscykeln

Vattenfallsmodellen är ett klassiskt exempel där denna metod tillämpas. Metoden består av en sekventiell serie av faser där varje fas har en samling aktiviteter som ska slutföras innan nästa fas påbörjas. Varje enskild fas har särskilda delresultat samt en detaljerad dokumentation som genomgått en grundlig granskningsprocess. (Mavuru, I., 2018)

2.2 Agila mjukvaruutvecklingsmetoder

Agila utvecklingsmetoder är baserade på iterativ utveckling. Denna arbetsmetod möjliggör det att arbeta med fungerande delleveranser regelbundet och att man löpande förbättrar planer och metoder i samtliga faser.

Den mest populära agila utvecklingsmetoden är Scrum som främst fokuserar på anpassningsförmåga, produktivitet och flexibilitet. Metoden är uppdelad i ett antal roller och arbetssteg där varje roll har olika uppgifter. Betydande roller är Scrum Manager och Product Owner. (Awad, 2005)

2.3 Betydande skillnader mellan traditionella och agila mjukvaruutvecklingsmetoder

Den huvudsakliga skillnaden mellan traditionella och agila mjuvaruutvecklingsmetoder är att arbetsflödet i den traditionella modellen är linjärt utformat, medan den agila modellen utgörs av ett iterativt arbetsflöde.

Traditionella metoder är lämpliga för projekt där kraven är specificerade och precisa; kunden vet precis vad de önskar att få ut av produkten och kan försäkra att det inte kommer ske omfattande förändringar under projektets utveckling. För större projekt där det finns ett stort utrymme för kontinuerlig modifiering anses traditionella metoder inte vara passande.

Projekt som involverar stora team och där frekventa kravändringar förväntas ske föredras det att man arbetar enligt en agil metodik, exempelvis Scrum. (Mavuru, I., 2018)

2.4 Parprogrammering

I det här kapitlet redogörs Extreme Programming och parprogrammering djupgående. Metoderna är presenterade i kapitel 2.4.1 och 2.4.2 i angiven ordning. kapitel 2.4.3 beskriver förarens roll i parprogrammering och kapitel 2.4.4 beskriver navigatörens roll.

2.4.1 Extreme Programming

Extreme Programming (XP) är en mjukvaruutvecklingsmetod som har för avsikt att förbättra mjukvarukvalitéten i ett projekt. Det är en agil utvecklingsmetod där det är rekommenderat att ha frekventa releaser under en kortvarig utvecklingslivscykel. Detta för att öka produktiviteten samt för att kunna introducera så kallade checkpoints som gör det möjligt för kunder att justera kravspecifikationen. En tillämpning av XP är parprogrammering. (Jeffries, 2011)

2.4.2 Definition av parprogrammering

Parprogrammering utförs av två programmerare som delar på en arbetsstation. De två rollerna som finns i denna arbetsprocess kallas förare och navigatör. Båda programmerarna är aktivt involverade i programmeringen men på två olika sätt: föraren skriver den faktiska koden på skrivbordet medan navigatören fokuserar till större del på den övergripande riktningen. Huvudsakligen utgår parprogrammering på att de två programmerarna skiftar roller frekvent, ofta var femtonde eller trettionde minut. (Williams, L. A. 2000)

2.4.3 Förarens roll

Föraren sköter sin del av arbetet genom att ha full kontroll över arbetsstationen: skrivbordet och musen. Rollen är avsedd att utföras på detta vis för att avlösa föraren från att fokusera på de taktiska aspekterna av att slutföra den pågående uppgiften och istället använda sig av navigatören som en säkerhetskontroll. För att nå arbetsmålet ska föraren åtgärda fel i koden och läsbarhetsproblem i skrivande stund, inte fokusera på större problem utan istället åtgärda dessa efter att ha programmerat färdigt. (Williams, L. A. 2000)

2.4.4 Navigatörens roll

Navigatörens arbetsuppgifter är att observera förarens arbete och att identifiera taktiska och strategiska brister. Större fokus ska läggas på att identifiera sätt att förenkla eller förbättra designen, fundera över större problem som kan uppstå och sammanställa de stora problem som finns. Navigatörens roll är ej avsedd för att diktera kod till föraren. När föraren är färdig med att koda ska navigatören ta upp de större problem som noterats under arbetets gång på ett artigt och sansat sätt. (Williams, L. A. 2000)

2.5 Förväntade fördelar

I detta kapitel beskrivs de fördelar som parprogrammering kan bidra med. Samtliga redogörelser är baserade på olika studier och experiment. Kapitel 2.5.1 redogör för fördelar i kodkvalitén medan kapitel 2.5.2 redogör för fördelar inom delad kunskap. Kapitel 2.5.3 tar upp hur parprogrammering bidrar till motivation hos utvecklare.

2.5.1 Kodkvalité

I ett experiment, dokumenterat av professor John T. Nosek, där 15 professionella programmerare arbetade om 5 par varav 5 soloprogrammerare med ett 45 minuters långt experiment (på ett verkligt och utmanande problem) redovisades lovande resultat för parprogrammering. De grupper som arbetat i par spenderade omkring 30 minuter för att slutföra uppgiften, vilket uppmättes vara 12 minuter snabbare än soloprogrammerarna. Läsbarheten i koden samt dess funktionalitet bedömdes för samtliga lösningar. I bedömningen framgår det att alla par överträffade samtliga soloprogrammerare. I sin studie dokumenterar Nosek även hur programmerarna själva upplevde uppgiften och hur den avsågs att lösas: i par eller enskilt. Av resultatet framgår det att de programmerare som arbetat i par med att lösa uppgiften, fattade ett bättre tycke för uppgiften än vad soloprogrammerarna gjorde. De programmerare som arbetat i par kände sig även mer säkra på sina slutliga lösningar, trots att de till en början hade varit skeptiska till att samarbeta i par. I sin slutsats sammanställer Nosek att parprogrammering möjliggör snabbare utveckling samt förhöjd mjukvarukvalitét. (Nosek, 1998)

I en studie av professor Laurie Williams och dataforskaren Alistair Cockburn där de använt sig av intervjuer och kontrollerade experiment till sitt förfogande, redogör de för samtliga kostnader och fördelar med parprogrammering. I deras studie framgår det att antalet buggar under parprogrammering minskar med cirka femton procent jämfört med soloprogrammering. (Cockburn & Williams, (2001)

2.5.2 Delade kunskaper

Parprogrammering medför möjligheten att ständigt dela med sig av sina kunskaper. Metoden driver de inblandade att ständigt byta roller som lärare och lärande genom att ge varandra olika tips, påpeka programmeringsspråksregler och olika designfärdigheter. (McDavid, S., 2019)

2.5.3 Motivation

Parprogrammering kräver att båda parter håller sig aktiva under hela arbetssessionen för att man ska kunna få ett pålitligt och bra resultat. Det är även viktigt att föraren "tänker högt" under tiden som denne programmerar. Detta för att det ska vara enkelt för navigatören att följa vad som försiggår och vilka avsikter som ligger bakom varje kodrad. På så sätt är båda lika insatta i uppgiften. Många tror att det "grupptryck" som uppstår när man samarbetar på det här viset driver motivationen hos samtliga programmerare till att slutföra uppgiften med större fokus och intensitet jämfört med soloprogrammerare. (Williams, Kessler, Cunningham, & Jeffries, 2000)

2.6 Förväntade nackdelar

Eventuella nackdelar som kan uppstå till följd av att arbeta med parprogrammering beskrivs i detta kapitel. Samtliga redogörelser är baserade på olika studier och experiment. Kapitel 2.6.1 redogör för de problem som uppstår när utvecklarna har olika färdighetsnivåer och förutsättningar och kapitel 2.6.2 tar upp vilka effekter metoden har på kostnaden för ett projekt.

2.6.1 Olika färdighetsnivåer och förutsättningar

Vid parprogrammering måste man väga upp gruppens kapacitet för att få ett så bra resultat av metoden som möjligt.

Något som kan framstå som problematiskt vid parprogrammering är att programmerarna har betydligt olika färdighetsnivåer. Dessutom kan effekten av parprogrammering skifta från person till person.

Det kan hända att två programmerare som uppskattas vara lika erfarna blir lidande för arbetet till följd av det så kallade "developer's ego phenomenon". Med detta menar man alltså att utvecklarna ständigt försöker överträffa varandra och är mer måna om att driva sina egna idéer i första hand.

Relationen mellan de två programmerarna är också avgörande för att arbetet ska gå smidigt till. I slutet av dagen går det inte att forcera människor att arbeta med varandra. (Williams, L., Kessler, R. R., Cunningham, W. & Jeffries, R., 2000)

2.6.2 Kostnader

Uppfattningen om att parprogrammering utgör en större kostnad för arbetsgivare genom att man i detta fall behöver betala ut ersättning till två programmerare är en vanlig nackdel som brukar tas upp i samband med utvecklingsmetoden. Det har dock gjorts ett flertal studier där just denna kostnadsfaktor undersöks. Värt att nämna återigen är Williams och Cockburns studie "The Costs and Benefits of Pair Programming" där de redogör för resultaten från ett experiment utfört vid universitetet University of Utah. I studien ifrågasätts denna uppfattning om högre kostnader då man tar hänsyn till de kostnader som undviks genom att arbeta i par; kodens kvalité och risken för buggar blir mindre vilket därav leder till lägre kostnader till att åtgärda defekter. (Cockburn & Williams, 2001)

2.7 Projektbeskrivning

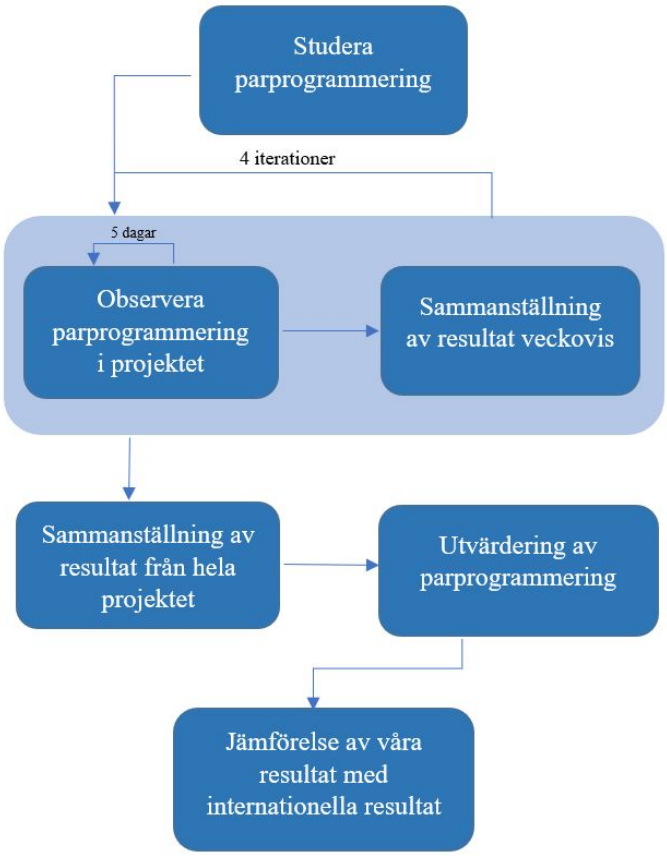
Projektet utförs under en period på fyra veckor i en grupp bestående av nio personer där vi använder den agila utvecklingsmetoden Scrum. Vår projektidé är en applikation som erbjuder studenter en plattform där de kan skapa flashcards, quizar och ladda upp anteckningar i syfte om att studera samt kunna dela material med andra studenter. Applikationen fungerar endast på operativsystemet Android och vi använder utvecklingsmiljön Android Studio samt programmeringsspråket Java för att utveckla applikationen.

3. Metod

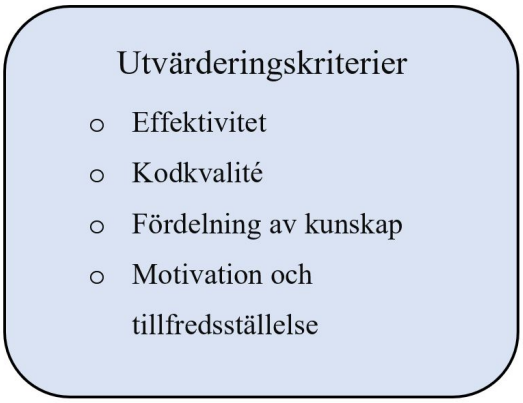
I detta kapitel beskrivs vår forskningsmetod. Kapitel 3.1 redogör för våra arbetssteg och kapitel 3.2 presenterar den utvärderingsmodell som användes.

3.1 Forskningsfaser

Vårt forskningsarbete bestod av sex faser. Som framgår av figur 2 är de följande: (1) Studera parprogrammering, (2) observera parprogrammering i projektet (3), sammanställning av resultat veckovis, (4) sammanställning av resultat från hela projektet, (5) utvärdering av parprogrammering och (6) jämförelse av våra resultat med internationella resultat. Nedan redogörs dessa faser.



Figur 2. Översikt av forskningsfaserna



Figur 3. Våra utvärderingskriterier

I första fasen studerade vi parprogrammering i syfte att få kunskap och förbereda oss för vår undersökning. Detta gjordes genom att söka efter och läsa litteratur som redogör för parprogrammering. Vi använde främst Google Scholar för att söka efter relevanta artiklar och böcker. Vi använde oss av nyckelorden: pair programming, software, methods, agile, traditional, Extreme Programming, benefits, disadvantages.

Kunskapen från de artiklar och böcker som vi studerat utgjorde basen för bakgrunden och introduktionen för vår undersökning. Första fasen bidrog till mycket kunskap om hur parprogrammering går till och varför man utför det. På så sätt kunde vi bestämma de utvärderingskriterierna som gällde för parprogrammering och skapa en modell att utgå ifrån när vi utvärderade parprogrammering.

Nästa fas bestod av att observera parprogrammering i projektet. Syftet var att dokumentera hur vår projektgrupp upplevde att parprogrammering fungerade i praktiken. Detta gjordes i fem dagar där varje dag avslutades med att våra gruppmedlemmar antecknade hur lång tid det tog att lösa uppgifter med parprogrammering och soloprogrammering. De fick också svara på några frågor som redogörs för i kapitel 3.2. Vid veckans slut sammanställdes alla resultaten i fas tre i syfte att ta reda på hur mycket tid som i genomsnitt ägnades åt att lösa uppgifter med parprogrammering samt om några brister upptäckts. Detta utfördes under fyra iterationer då projektet pågick i fyra veckor.

När projektet var slut sammanställde vi all dokumentation från hela projektet och analyserade våra resultat. Tiden som ägnades åt varje uppgift summerades ihop. Dessa resultat använde vi sedan i fas fem under utvärderingen av parprogrammering i syfte att kunna identifiera brister. I den sista fasen jämförde vi samtliga resultat med internationella studiers resultat. Detta i syfte till att bilda en god uppfattning om hur slutsatserna i denna rapport förhåller sig i relation till övriga forskningsresultat.

3.2 Utvärderingsmodellen

För att kunna utvärdera parprogrammering måste man ha flera faktorer i åtanke. Som framgår av figur 3 består vår utvärderingsmodell av 4 olika kriterier där alla beaktas ur både ett kortsiktigt och långsiktigt perspektiv. Dessa är följande:

- *Effektivitet:* Tidigare studier har visat att parprogrammering bidrar till att uppgifter löses på färre arbetstimmar i jämförelse med soloprogrammering. Däremot anser många företag att parprogrammering är en tidskritisk resurs. De menar att arbetsåtgången ökar till det dubbla eftersom man sätter två programmerare på ett arbete som en programmerare kan göra. (Williams et al., 2000) Med anledning av den betydande skillnaden i dessa slutsatser fann vi det intressant att undersöka om parprogrammering är mer effektivt än soloprogrammering. Undersökningen gick till på så sätt att våra gruppmedlemmar fick dokumentera den totala tiden som ägnades åt att lösa uppgifter med parprogrammering och soloprogrammering. Efter varje sprint samlade vi in alla tidrapporteringar.
- *Kodkvalité:* Huvudargumentet för parprogrammering är dess ökning på kodkvalitén. Med bättre kodkvalité menar man att koden som skrivs har bättre struktur och design samt att den är mer korrekt och stabil. (Cockburn & Williams, 2001) Med anledning av detta tyckte vi att det var självklart att detta kriterium var nödvändigt för vår undersökning. Vi undersökte huruvida kodkvalitén förbättrades eller inte med parprogrammering genom att vi jämförde struktur, design och hur felfri och stabil koden var i jämförelse med en enskild programmerare.

- *Fördelning av kunskap:* Med parprogrammering fördelas kunskap mellan föraren och navigatören, såsom teknik, algoritmer och generella tips (Cockburn & Williams, 2001). Vi undersökte om våra gruppmedlemmar fått några nya kunskaper av

varandra efter att ha använt parprogrammering genom att de under projektets gång fick svara på några frågor.

Frågorna som ställdes var följande.

1. Anser du att parprogrammering möjliggjort enklare fördelning av kunskaper och kompetenser i gruppen?
2. Tror du att du hade löst problemet minst lika bra om du antog dig problemet som soloprogrammerare?
3. Tycker du att du har utvecklats som programmerare efter att ha utfört parprogrammering? Varför?

- *Motivation och tillfredsställelse:* Programmerare har uttryckt att det är roligare och att de blir mer motiverade när de jobbat i par. Den största anledningen är att man då kan diskutera idéer och att man inte vill svika sin partner. (Williams et al., 2000) Vi undersökte om våra gruppmedlemmar kände att de blev mer motiverade och tillfredsställda av att jobba i par jämfört med enskilt. Undersökningen gick ut på att våra gruppmedlemmar fick svara på några frågor rörande motivation och tillfredsställelse vid parprogrammering.

  Frågorna som ställdes var följande.

  1. Var det jobbigt att utföra parprogrammering med någon som hade mindre erfarenhet inom programmering?
  2. Var det jobbigt att utföra parprogrammering med någon som hade mer erfarenhet inom programmering?
  3. Hur tycker du att parprogrammering påverkar utvecklingsmiljön?

4. Resultat

I detta kapitel redovisas resultaten för vår utvärdering av parprogrammering. Vi följer utvärderingsmodellens ordning där varje kriterium presenteras i ett enskilt avsnitt. Dessa presenteras i kapitel 4.1-4.4.

4.1 Effektivitet

Vi undersökte den totala tiden som ägnades åt att lösa en uppgift med parprogrammering och soloprogrammering i syfte att ta reda på vilket arbetssätt som var mer effektivt.

I vårt projekt var det inte tidseffektivt att jämföra parprogrammering med soloprogrammering på samma programmeringsuppgifter då det skulle leda till förseningar under sprintsen. Istället undersökte vi effektiviteten på de uppgifter som vi uppskattade till att ta ungefär lika lång tid att lösa.

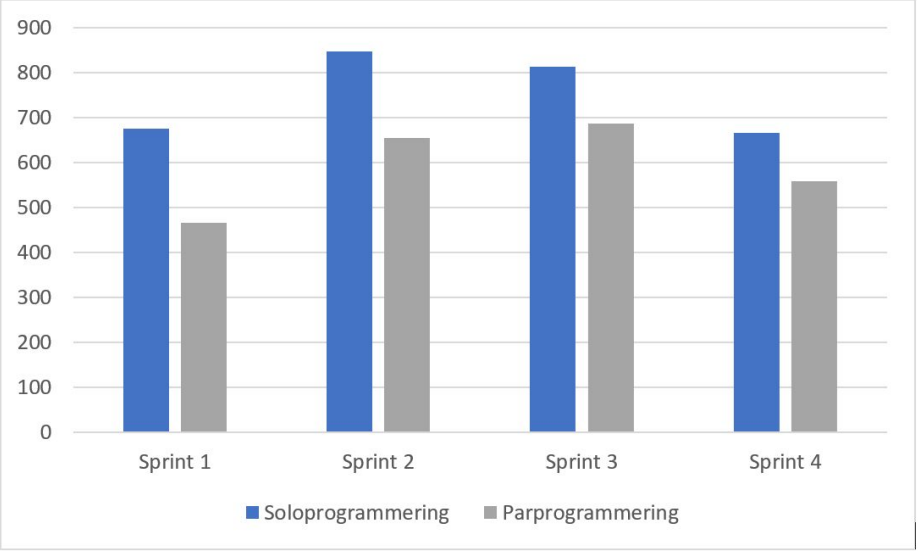
Vi har delat upp dessa uppgifter i tre kategorier: mindre tidskrävande uppgifter, någorlunda tidskrävande uppgifter och mycket tidskrävande uppgifter. I tabell 1 och 2 visas tiden det tog i genomsnitt att utföra en uppgift med parprogrammering och soloprogrammering. I figur 4 illustreras medeltidsförloppet för soloprogrammering i relation till parprogrammering.

Tabell 1. Medelvärdet av alla tidrapporteringar för parprogrammering

Parprogrammering	Mindre tidskrävande uppgifter (<30 min)	Någorlunda tidskrävande uppgifter (30< min <8h)	Mycket tidskrävande uppgifter (>8h)
Sprint 1	16,4	396,4	983
Sprint 2	11	234,2	1718,3
Sprint 3	19,6	468,6	1571,2
Sprint 4	16	649	1008,7

Tabell 2. Medelvärdet av alla tidrapporteringar för soloprogrammering

Soloprogrammering	Mindre tidskrävande uppgifter (<30 min)	Någorlunda tidskrävande uppgifter (30< min <8h)	Mycket tidskrävande uppgifter (>8h)
Sprint 1	16,8	472	1536
Sprint 2	12	325,4	2203,2
Sprint 3	20,7	526,6	1893
Sprint 4	18	637	1345



Figur 4. Medeltidsförloppet för soloprogrammering i relation till parprogrammering

4.2 Kodkvalité

Vi undersökte om parprogrammering ledde till en förbättrad kodkvalité. Genom att jämföra kodkvalitén på kod skriven av parprogrammerare med kod skriven av soloprogrammerare i fyra veckor kunde vi komma fram till några tydliga resultat som vi rapporterar nedan i tabell 3.

Tabell 3. Resultat från vår observation av kodkvalitén

	Struktur	Design	Fel	Stabilitet
Parprogrammering	Färre rader kod, återanvändbar kod, korta if-satser, korta metoder.	Tydliga variabel- och metodnamn, oftast välskrivna kommentarer, korta parameterlistor	Oftast felfri.	Stabil
Soloprogrammering	Fler rader kod, ej återanvändbar kod, långa metoder, ibland långa if-satser	Oklara variabel- och metodnamn, saknar ofta kommentarer, ibland oläslig kod, långa parameterlistor	Några små fel	Oftast stabil

4.3 Fördelning av kunskap

I syfte att ta reda på om parprogrammering bidrar till fördelning av kunskap mellan navigatören och föraren undersöktes våra gruppmedlemmars personliga åsikter genom att de fick svara på några frågor under projektets gång. Några citat som representerar de vanligaste svaren presenteras i tabell 4.

I svaren till fråga 1 kunde vi konstatera att majoriteten av våra gruppmedlemmar ansåg att parprogrammering bidrog till enklare fördelning av kunskaper och kompetenser i gruppen. På fråga 2 var gruppmedlemmarna överens om att parprogrammering bidrog till en bättre lösning när det gällde svårare uppgifter. Typiska kommentarer på fråga 2 var "Små uppgifter hade jag kunnat lösa minst lika bra" och "Det hade varit fulare kod och flera fel". Även på fråga 3 var majoriteten av gruppmedlemmarna eniga om att de utvecklats som programmerare till följd av parprogrammering.

Tabell 4. Typsvar från frågorna om fördelning av kunskap

Frågor	Typsvar 1:	Typsvar 2:
1	"Ja, om man får jobba med någon som kan mer eller är lika duktig"	"Ja, oftast är man bra på olika områden, då kunde man lära sig av varandra"
2	"Nej, det hade nog varit fulare kod och flera fel"	"Beror på vilken uppgift. Små uppgifter hade jag kunnat lösa minst lika bra, men med större uppgifter är det bra att ha någon som man kan reflektera med"
3	"Ja, eftersom man hela tiden diskuterar och bollar idéer så lär man sig mycket"	"Ja, till följd av att man kontinuerligt byter roller mellan att vara navigatör och förare så lär man sig av varandra"

4.4 Motivation och tillfredsställelse

Vi undersökte om parprogrammering ledde till mer motiverade och tillfredsställda programmerare genom att våra gruppmedlemmar fick svara på några frågor under projektets gång. Nedan presenteras några citat från svaren som representerar de vanligaste svaren som erhölls.

Utifrån typsvaren kan man konstatera att våra gruppmedlemmarna var ganska överens om fråga 1 och 2. På fråga 3 var det däremot en tydlig splittring i svaren, några tyckte att utvecklingsmiljön förbättrades av parprogrammering medan resterande tyckte att det försämrade utvecklingsmiljön. Typiska kommentarer på fråga 3 var "Det berodde på vem man fick jobba med".

Tabell 5. Typsvar från frågorna om motivation och tillfredsställelse

Frågor	Typsvar 1:	Typsvar 2:
1	"Ja, det kunde bli väldigt frustrerande att behöva förklara hela tiden"	"Ibland, men man lär sig mycket själv också när man måste förklara för någon annan"
2	"Ja, det kändes som att jag inte tillförde så mycket"	"Ibland, det var svårt att hänga med men samtidigt så lärde jag mig en del"
3	"Bättre eftersom man hjälper varandra och det blir roligare än att sitta själv, särskilt i vårt fall då vi jobbade hemifrån"	"Dåligt, berodde mycket på vem man fick jobba med. Tror jag hade jobbat på bättre själv, men jobbar man med någon som är lika duktig så hade det varit roligt att kunna bolla idéer"

5. Analys

Utifrån tabell 1 och tabell 2 kan man konstatera att det tar färre arbetstimmar för parprogrammerare att lösa uppgifter som är någorlunda och/eller mycket tidskrävande jämfört med soloprogrammerare. Däremot påvisas inga större skillnader när det kommer till mindre tidskrävande uppgifter. Detta samspelar relativt väl med de resultat som det redogjorts för i rapportens bakgrund. Ur ett arbetsgivarperspektiv är det ytterst lönsamt att väga vilka faktorer som har störst inverkan innan man beslutar om att införa parprogrammering som utvecklingsmetod. Trots att denna rapport stödjer de underlag som finns för att parprogrammering stärker kodkvalitén samtidigt som den bidrar till hög produktivitet genom arbetsprocessen, är det av stor vikt att man som arbetsgivare väger upp kostnadseffektiviteten som parprogrammering medför.

Utifrån empiriska erfarenheter samt de resultat som redovisats i tabell 4 går det att dra en slutsats om att programmerarna själva upplever en förbättrad kodkvalité och ökad läsbarhet av koden när uppgifterna genomförts med hjälp av parprogrammering. Vidare gällande motivationen hos programmerarna och upplevd tillfredsställning har varierande resultat uppmätts. De presenterade resultaten i kapitel 4.4 bekräftar de förväntade för- och nackdelar som angivits i rapportens bakgrund (se kapitel 2.5 och 2.6). Huruvida det går att effektivisera parprogrammering så mycket som möjligt är beroende av många olika faktorer. I bästa fall kommer programmerarna väl överens med varandra samtidigt som de ser till att utnyttja de bästa kompetenserna hos varandra. Som presenterat i tabell 5 upplever programmerare väldigt positiva resultat av parprogrammering när de trivs med sin arbetskollega samtidigt som de kan dra stor nytta av varandras kompetenser. Intressant att notera är att genom att arbeta i par, upplevs stimulansen öka hos samtliga programmerare för att diskutera flera olika idéer som resulterar i nya, mer effektiva lösningar. Ur ett utvecklingsperspektiv är detta ett väldigt positivt utfall då tekniken i dagens samhälle ständigt strävar efter nya, innovativa och effektiva lösningar.

Parprogrammering fallerar dock när programmerare uppskattas ha stora kompetensskillnader sinsemellan vilket ofta resulterar i att uppgiften tar längre tid eller i att kemin mellan programmerarna inte fungerar. Kompetensskillnader mellan teammedlemmar var något som vi upplevde påverkade arbetet till större del. När man

arbetade med någon som var mindre kunnig inom ett visst område eller kände sig osäker var det svårt att strukturera upp arbetet. En stor del av tiden gick åt att vägleda denne vilket i sin tur bidrog till att arbetet inte blev särskilt effektivt. Det dröjde ut på arbetsprocessen samtidigt som det var ganska omständigt att inte riktigt kunna strukturera upp och fördela arbetet. Även om utbyte av kunskaper mellan arbetskollegor har sina fördelar upplevde vi detta som en nackdel med parprogrammering då det enligt denna studie beror väldigt mycket på vem man arbetar med. Baserat på våra resultat från tabell 5 kan det inte fastställas huruvida parprogrammering leder till mer motiverade och tillfredsställda programmerare. En relativt rimlig slutsats är att parprogrammering, vid "rätt" val av partners, kan leda till ökad motivation och tillfredsställelse hos samtliga programmerare. Ett förslag till förbättring är att arbetsgivaren under projektets gång bör skaffa sig en uppfattning om vilka utvecklare som jobbar bra tillsammans och därmed föreslå lämpliga par. På så sätt ser man till att kompetenser mellan utvecklare fördelas på bästa sätt samtidigt som arbetet blir mer effektivt.

Kostnaden för detta projekt gällande nyttjad arbetskraft i relation till utförda arbetstimmar har inte kunnat analyseras eftersom projektet i II1305 utfördes i rent utbildningssyfte. Det som dock styrkts i den studie som nämnts i kapitel 2.6.2 är att parprogrammering medför kostnadseffektivitet på sådant sätt att risken minskas för att behöva åtgärda större defekter och brister i koden som skulle kunna leda till stora utgifter. Detta kunde vi och övriga medlemmar i vårt team också konstatera, då det hela tiden var många som bearbetade koden regelbundet. Det var enklare att analysera samt felsöka koden tillsammans med andra programmerare eftersom alla bidrog med olika kompetenser inom olika områden.

De nackdelar som det redogjorts för i kapitel 2.6 bekräftas av vår empiriska erfarenhet inom projektarbetet. Vi finner att parprogrammering är en väldigt effektiv metod samtidigt som det är väldigt krävande. Metoden underlättar till viss del arbetsprocessen eftersom man känner sig säkrare på sina lösningar när de formulerats och utvecklats i sällskap av ytterligare en programmerare.

6. Slutord

Trots att många studier visar märkbara fördelar med parprogrammering vet vi lite om hur gynnsam metoden är effektivitetsmässigt. I den här rapporten utvärderade vi parprogrammering med utgångspunkt från vårt projekt. Vårt mål var att få en ökad förståelse om hur parprogrammering fungerar i praktiken och därmed föreslå förbättringar för framtida projekt.

Det som konstaterats i analysen är i huvudsak att man måste väga in flera olika faktorer innan man beslutar om att inkorporera parprogrammering som utvecklingsmetod för ett projekt. En möjlig implementation av parprogrammering är att man gör en god tidsuppskattning för samtliga uppgifter inom ett projekt och utifrån dessa avgör var man har störst nytta av att implementera parprogrammering. På så sätt kan man effektivisera parprogrammering.

Eftersom utvecklare både är en dyr och svårersättlig resurs är det viktigt att företag satsar på att deras anställda uppskattar och trivs i sin arbetsmiljö för att kunna behålla dem. Då vi utifrån våra resultat kan konstatera att utvecklare överlag uppskattar parprogrammering som utvecklingsmetod kan det vara gynnsamt för företag att satsa på det.

Metoden har endast prövats under en period på fyra veckor i utbildningssyfte vilket innebär att våra resultat inte är applicerbart när det kommer till projekt i företag. Den slutsatsen som går att dra är att parprogrammering fungerat väl i vår miljö men att fler studier krävs för att utvärdera parprogrammering och effekten av våra förbättringar.

7. Referenser

Awad, M. A. (2005). *A comparison between agile and traditional software development methodologies*. Hämtad April 17, 2020, från University of North Floridas webbplats: https://www.unf.edu/~broggio/cen6940/ComparisonAgileTraditional.pdf.

Cockburn A., & Williams, L. (2001). *The Costs and Benefits of Pair Programming.* Hämtad Maj 15, 2020 från University of Michigans webbplats: https://web.eecs.umich.edu/~weimerw/481/readings/pairprogramming.pdf.

Jeffries, R. (2011). *What is Extreme Programming?* Hämtad Maj, 28, 2020 från https://ronjeffries.com/xprog/what-is-extreme-programming/.

Mavuru, I. (2018) *Traditional vs. Agile Software Development Methodologies.* Hämtad Maj, 26, 2020 från https://www.kpipartners.com/blog/traditional-vs-agile-software-development-methodologies

McDavid, S. (2019). *The Pros and Cons of Pair Programming.* Hämtad Maj, 27, 2020 från https://www.verypossible.com/blog/pros-and-cons-of-pair-programming.

Nilsson, K. (2003). *Increasing Quality with Pair Programming.* Hämtad Maj, 4, 2020, från https://www.diva-portal.org/smash/get/diva2:831281/FULLTEXT01.pdf.

Nosek, J. T. (1998) The Case for Collaborative Programming. Hämtad Maj, 27, 2020 från https://www.researchgate.net/publication/27295641_The_Case_for_Collaborative_Programming.

Williams, L. A. (2000). *The Collaborative software process.* Hämtad Maj, 4, 2020 från https://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf.

Williams, L. A., Kessler, R. R., Cunningham, W. & Jeffries, R. (2000). *Strengthening the Case for Pair-Programming*. *IEEE Software, 17*(4), 19-25. https://doi.org/10.1109/52.854064.

8. Bilagor

Tabell 1. Medelvärdet av alla tidrapporteringar för parprogrammering

Parprogrammering	Mindre tidskrävande uppgifter (<30 min)	Någorlunda tidskrävande uppgifter (30< min <8h)	Mycket tidskrävande uppgifter (>8h)
Sprint 1	16,4	396,4	983
Sprint 2	11	234,2	1718,3
Sprint 3	19,6	468,6	1571,2
Sprint 4	16	649	1008,7

Tabell 2. Medelvärdet av alla tidrapporteringar för soloprogrammering

Soloprogrammering	Mindre tidskrävande uppgifter (<30 min)	Någorlunda tidskrävande uppgifter (30< min <8h)	Mycket tidskrävande uppgifter (>8h)
Sprint 1	16,8	472	1536
Sprint 2	12	325,4	2203,2
Sprint 3	20,7	526,6	1893
Sprint 4	18	637	1345

Tabell 3. Resultat från vår observation av kodkvalitén

	Struktur	Design	Fel	Stabilitet
Parprogrammering	Färre rader kod, återanvändbar kod, korta if-satser, korta metoder.	Tydliga variabel- och metodnamn, oftast välskrivna kommentarer, korta parameterlistor	Oftast felfri.	Stabil
Soloprogrammering	Fler rader kod, ej återanvändbar kod, långa metoder, ibland långa if-satser	Oklara variabel- och metodnamn, saknar ofta kommentarer, ibland oläslig kod, långa parameterlistor	Några små fel	Oftast stabil

Tabell 4. Typsvar från frågorna om fördelning av kunskap

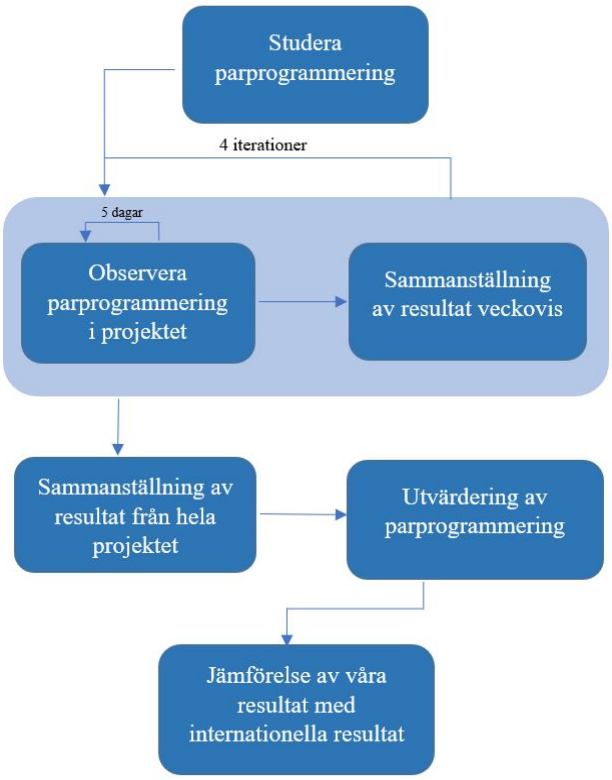
Frågor	Typsvar 1:	Typsvar 2:
1	"Ja, om man får jobba med någon som kan mer eller är lika duktig"	"Ja, oftast är man bra på olika områden, då kunde man lära sig av varandra"
2	"Nej, det hade nog varit fulare kod och flera fel"	"Beror på vilken uppgift. Små uppgifter hade jag kunnat lösa minst lika bra, men med större uppgifter är det bra att ha någon som man kan reflektera med"
3	"Ja, eftersom man hela tiden diskuterar och bollar idéer så lär man sig mycket"	"Ja, till följd av att man kontinuerligt byter roller mellan att vara navigatör och förare så lär man sig av varandra"

Tabell 5. Typsvar från frågorna om motivation och tillfredsställelse

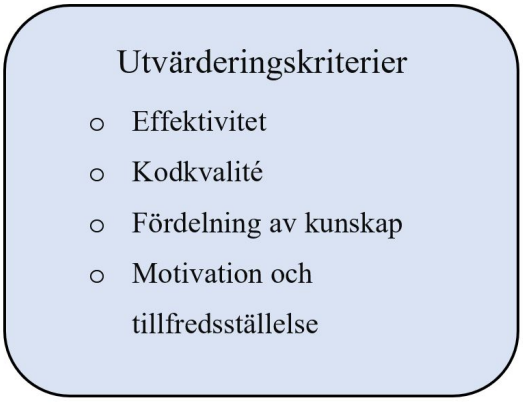
Frågor	Typsvar 1:	Typsvar 2:
1	"Ja, det kunde bli väldigt frustrerande att behöva förklara hela tiden"	"Ibland, men man lär sig mycket själv också när man måste förklara för någon annan"
2	"ja, det kändes som att jag inte tillförde så mycket"	"Ibland, det var svårt att hänga med men samtidigt så lärde jag mig en del"
3	"Bättre eftersom man hjälper varandra och det blir roligare än att sitta själv, särskilt i vårt fall då vi jobbade hemifrån"	"Dåligt, berodde mycket på vem man fick jobba med. Tror jag hade jobbat på bättre själv, men jobbar man med någon som är lika duktig så hade det varit roligt att kunna bolla idéer"

Figur 1. Mjukvaruutvecklingslivscykeln

Figur 2. Översikt av forskningsfaserna



Figur 3. Våra utvärderingskriterier



Figur 4. Medeltidsförloppet för soloprogrammering i relation till parprogrammering