HÖGSKOLAN DALARNA

Utforskning av styrkor och svagheter med par-, mobb- och soloprogrammering inom nyutveckling och förvaltning: Utvecklares perspektiv

En fallstudie på Trafikverket

Exploring the strengths and weaknesses of pair, mob and solo programming in new development projects and maintenance: Developers perspective

Författare: Simon Kvarnström & Jesse Vähäjylkkä

Handledare: Ulrika Artursson Wissa
Examinator: Rikard Land

Kurskod – GIK28T
Examensarbete för filosofie kandidatexamen i Informatik
2023-05-16
Publicerad i fulltext, fritt tillgängligt

Sammanfattning

Bakgrund	Parprogrammering och mobbprogrammering är samarbetsinriktade programutvecklingstekniker där utvecklare arbetar tillsammans för att lösa problem. Fördelarna inkluderar förbättrad kodförståelse, minskad defektfrekvens och kunskapsöverföring. Mobbprogrammering involverar en hel grupp av programmerare som samarbetar med samma kod. Båda metoderna leder till ökad produktivitet och kunskapsbreddning bland utvecklarna. Dessa metoder undersöktes i samarbete med Trafikverket för att förbättra förståelsen av deras effektivitet inom olika utvecklingskontexter.
Syfte	Vilka styrkor och svagheter upplever utvecklare med par-, mobb- och soloprogrammering inom nyutveckling och förvaltning?
Metod	Datainsamlingen bestod av semistrukturerade intervjuer och en enkät. Urvalet för intervjuerna gjordes av vår handledare på Trafikverket, som valde personer med erfarenhet av parprogrammering. (Enkäten skickades ut till 205 utvecklare och vi fick 23 svar, vilket motsvarar en svarsfrekvens på 11,2%.)
Analys	Resultatet visar att utvecklare hade störst preferens för att programmera ensam i nyutveckling (43%) samt i förvaltning (35%). Undersökningen visade även att (82%) ser kompetensspridning som en anledning att använda sig av par- eller mobbprogrammering.
Slutsatser	Styrkorna med soloprogrammering är att det oftast går snabbare att utveckla, utvecklaren kan jobba mer självständigt och efter sina egna preferenser. Styrkorna med par- och mobbprogrammering är att kodkvaliteten ökar och förmågan att sprida kunskap. Oftast går det snabbare att lösa problem i par och mobb. Svagheterna med par- och mobbprogrammering är att det krävs fler resurser och det kan ta längre tid att göra framsteg.
Följder	Resultatet av denna undersökning kan vara till nytta för organisationer som använder eller överväger att använda par- eller mobbprogrammering som metod för utveckling, och bidra till att optimera deras utvecklingsprocesser.
Nyckelord	Parprogrammering, Mobbprogrammering, Soloprogrammering

Abstract

Background	Pair programming and mob programming are collaborative software development techniques where developers work together to solve development problems. The benefits include improved code understanding, reduced defect rates, and knowledge transfer. Mob programming involves an entire team of programmers collaborating on the same code. Both techniques lead to increased productivity and knowledge sharing among developers. These techniques were investigated in collaboration with Trafikverket to improve understanding of their effectiveness in different development contexts.
Aim	To find out what strengths and weaknesses developers experience with pair programming, mob programming and solo programming in the contexts of new development projects and maintenance.
Method	The data collection involved semi-structured interviews and a survey. The selection for the interviews was done by our supervisor at Trafikverket, who chose individuals with experience in pair programming. The survey was distributed to 205 developers, and we received 23 responses, resulting in a response rate of 11.2%.
Analysis	The result shows that developers had the greatest preference for programming alone in new development (43%) and in maintenance (35%). The survey also showed that (82%) see the spread of skills as a reason to use pair or mob programming.
Conclusions	The strengths of solo programming include faster development, the ability for developers to work independently and according to their own preferences. The strengths of pair and mob programming include improved code quality and knowledge sharing. Problem-solving is often faster in pair and mob programming. The

	weaknesses of pair and mob programming include the need for more resources and potentially longer time to make progress.
Implications	The results of this study can be beneficial for organizations that currently use or are considering adopting pair programming or mob programming as a development method. It can contribute to optimizing their development processes and improving their overall outcomes.
Keywords	Pair programming, Mob programming, Solo programming

Förkortningar och definitioner

Parprogrammering	"Parprogrammering (pair programming på engelska) är en programutvecklingsteknik där två programmerare jobbar tillsammans vid en gemensam dator" Källa: https://www.csc.kth.se/tcs/projects/cerise/parprogrammering/
Föraren	Programmeraren som aktivt skriver koden vid den gemensamma datorn. Källa: https://www.csc.kth.se/tcs/projects/cerise/parprogrammering/
Navigatören	Programmeraren som granskar koden som skrivs av föraren. Källa: https://www.csc.kth.se/tcs/projects/cerise/parprogrammering/
Mobbprogrammering	Mobbprogrammering är en samarbetsinriktad metod inom mjukvaruutveckling där ett team arbetar tillsammans på en enda dator. Källa: Zuill och Meadows (2016)
Soloprogrammering	Soloprogrammering är när en utvecklare arbetar individuellt för att programmera en specifik uppgift. Källa: https://www.linkedin.com/advice/1/how-do-you-measure-quality-productivity

Innehåll

1 Introduktion

1.1 Bakgrund

Inom programvaruutveckling finns det olika metoder och arbetssätt som används för att genomföra utvecklingsprojekt. Bland dessa metoder framträder par-, mobb- och soloprogrammering som tre betydande tillvägagångssätt. Soloprogrammering innebär att en enskild utvecklare tar på sig hela ansvaret för både design och kodskrivning.

Parprogrammering är en samarbetsinriktad programutvecklingsteknik som introducerades som en av de tio principerna inom Extreme Programming. Metoden innebär att två programmerare arbetar tillsammans vid en gemensam dator. Den ena agerar som förare (Engelska: driver) och den andra som navigatör (Engelska: navigator). Föraren skriver kod aktivt medan navigatören noggrant observerar och granskar koden som föraren skriver. (Sun et al., 2016)

Två huvuden är bättre än ett är ett vanligt uttryck som hänvisar till fördelarna med samarbete. Värdet av samarbete uppmuntras uttryckligen i mjukvaruutveckling genom en praxis känd som parprogrammering. Parprogrammering är en mjukvaruutvecklingsteknik där två utvecklare arbetar nära tillsammans för att lösa ett utvecklingsproblem. Några fördelar med parprogrammering har lyfts fram, inklusive förbättrad förståelighet och underhållbarhet av kod och design, minskad defektfrekvens och kunskapsöverföring. Med tanke på att utvecklare aldrig har identisk kunskap, kan en viss grad av kunskapsöverföring också förväntas inom varje parprogrammering konstellation. (Plonka et al., 2015)

De flesta mjukvaruutvecklingsteam är sammansatta av utvecklare med olika kunskapsnivåer, inklusive olika programmeringserfarenhet, olika domänexpertis och kunskap om olika tekniker. Parprogrammering är ett sätt att dela sin kunskap med andra utvecklare och samtidigt uppnå meningsfullt arbete. I vissa fall är kunskapsöverföring det uttryckliga målet för en session av parprogrammering. Detta är vanligt när en mer erfaren utvecklare lär en mindre erfaren utvecklare för att t.ex. få fart på ny personal. (Plonka et al., 2015)

Mobbprogrammering är en programutvecklingsteknik där en hel grupp av programmerare samarbetar med samma kod. Till skillnad från parprogrammering, inkluderar mobbprogrammering fler än en navigatör medan det fortfarande används en enda dator för att skriva kod. (Zuill & Meadows, 2016) Enligt Zuill och Meadows (2016) är mobbprogrammering en evolutionär utveckling av konceptet parprogrammering inom Extreme Programming.

Enligt Jim Buchan (2018) har utvecklare blivit mer konsekventa i sitt tillvägagångssätt för kodning och design genom mobbprogrammering. Det har skapat en konsekvent kodstil i hela gruppen. Utvecklarna har också blivit mer konsekventa i användningen av verktyg för utveckling och mer effektiva i sin användning av verktygen. Detta har ökat produktiviteten då verktygen integreras smidigare i arbetet. Genom mobbprogrammering har utvecklarna breddat sin kunskap om systemet jämfört med deras ursprungliga specialiserade område. Tidigare fanns en tydlig fördelning av kunskap och färdigheter mellan utvecklare i front-end och back-end. Efter har alla utvecklare byggt tillräckligt med kompetens att arbeta med både front-end och back-end. Fördelarna med detta är att det blir enklare att fördela arbetet och det minskar beroendet av en specifik utvecklare med kunskap inom ett område av systemet.

Denna undersökning genomfördes i samarbete med Trafikverket. Myndigheten valde att undersöka användningen av par-, mobb- och soloprogrammering, för att förbättra förståelsen av effektiviteten i olika utvecklingskontexter. Trafikverket är en svensk myndighet som ansvarar för planering, byggande och underhåll av infrastrukturen för väg, järnväg, sjöfart och luftfart i Sverige. Deras vision är att skapa en hållbar och tillgänglig transportinfrastruktur för alla. Trafikverket har ett brett uppdrag som sträcker

sig från att säkerställa trafiksäkerheten till att minska miljöpåverkan och främja samhällsutvecklingen. (Trafikverket, 2023)

1.2 Problembeskrivning

En svår fråga inom programmering är att avgöra vilken typ av programmering – parprogrammering, mobbprogrammering eller soloprogrammering – som utvecklare anser vara mest effektiv vid skapande av ny kod eller vid vidareutveckling av befintlig kod (förvaltning).

Genom att utforska och förstå vilken metod utvecklare upplever som mest effektiv i olika utvecklingskontexter, kan resultatet av denna undersökning bidra till en optimerad användning av par-, mobb- och soloprogrammering. Detta kan bidra till bättre arbetsflöden inom utveckling samt beslutsfattande kring val av par-, mobb- och soloprogrammering.

Utifrån beskrivningen kommer följande fråga besvaras:

- Vilka styrkor och svagheter upplever utvecklare med par-, mobb- och soloprogrammering inom nyutveckling och förvaltning?

1.3 Syfte

Syftet med denna undersökning är att jämföra upplevd effektivitet mellan par-, mobb- och soloprogrammering, både vid utveckling av ny kod och vid vidareutveckling av befintlig kod. Undersökningen avser att bidra till en fördjupad förståelse för vilken metod som kan vara mer effektiv i olika utvecklingskontexter. Genom att erhålla insikter i utvecklarnas upplevelser av par-, mobb- och soloprogrammering kan vi få en mer holistisk bild. Detta kan bidra till en diskussion om bästa praxis vid val av par-, mobb- eller soloprogrammering som tillvägagångssätt vid olika utvecklingsprojekt.

1.4 Avgränsningar

Undersökningens intervjuer är begränsade till Trafikverkets huvudkontor i Borlänge och omfattar inte andra kontor eller avdelningar inom organisationen. Enkäten är öppen för respondenter från olika enheter inom Trafikverket och inte begränsad till ett specifikt kontor. Det är viktigt att påpeka att denna undersökning inte inkluderar mätningar av faktorer såsom effektivitet eller tidsåtgång.

2 Teori

2.1 Soloprogrammering

Enligt Williams och Kessler (2000) ansåg hälften av deltagarna i deras studie att det var acceptabelt att arbeta ensamma 10-50% av tiden. Många föredrar att utföra experimentella prototyper, lösa problem som kräver djup koncentration och genomföra logiskt tänkande på egen hand. De flesta deltagare i studien höll med om att enkel och väldefinierad rutinkodning utförs mer effektivt av en ensam programmerare, medan en partner kan granska koden efteråt.

2.2 Effektivitetsutvärdering av parprogrammering: Fördelar och nackdelar

I en genomförd studie (Sun et al., 2016) utvärderades effektiviteten hos parprogrammering jämfört med soloprogrammering. Forskarna mätte de upplevda fördelarna och nackdelarna med avseende på fyra variabler:

- *Ansträngning (Effort)* – antalet persontimmar som krävdes för att slutföra ett projekt.
- *Defektfrekvens (Defect rate)* - antalet defekter per tusen rader kod (KLOC).
- *Kunskapsöverföring (Knowledge transfer)* - kommunikation av kunskap från en källa så att en mottagare lär sig och tillämpar den.
- *Kostnad (Cost)* - projektets totala kostnad.

Enligt respondenterna med erfarenhet av parprogrammering ansåg de att parprogrammering minskar projektets totala kostnad med 12 procent. Respondenterna som saknade erfarenhet av parprogrammering ansåg istället att parprogrammering ökar kostnaden med 5 procent. Båda grupperna ansåg att parprogrammering ökar ansträngningen i projekt med låg komplexitet, men minskar i projekt med hög komplexitet. Båda grupperna var överens om att parprogrammering minskar defektfrekvensen i projekt oavsett komplexitetsnivå, och att defektfrekvensen minskar mer ju högre projektets komplexitet är. Däremot skilde sig grupperna åt när det gällde minskningens omfattning av defektfrekvensen. Båda grupperna var även överens om att parprogrammering leder till ökad kunskapsöverföring och att mer kunskapsöverföring sker ju högre projektets komplexitet är. Även här skiljer sig grupperna åt i den förväntade omfattningen av kunskapsöverföring. Båda grupperna ansåg att par bestående av juniora utvecklare ökar ansträngningen, medan par bestående av seniora utvecklare minskar ansträngningen. Respondenterna med erfarenhet av parprogrammering ansåg att par bestående av en junior och en senior utvecklare minskar ansträngningen. Den andra gruppen trodde att sådana par ökar ansträngningen. (Sun et al., 2016)

Många människor avvisar idén om parprogrammering eftersom de antar att att sätta två programmerare på ett jobb som en person kan göra kommer att fördubbla arbetsbelastningen. Dock visade experimentet i Williams et al. (2000) att efter den inledande omställningstiden minskade de totala arbetsinsatserna dramatiskt för varje uppgift. Studien kom fram till att genom att arbeta i par kunde paren slutföra sina uppgifter mellan 40 och 50% snabbare.

I en studie av Lui och Chan (2006) framkom två principer relaterade till parprogrammering. Den första principen visade att ett par var betydligt mer produktivt när det kom till att slutföra uppgifter inom en kortare tidsram. Dessutom kunde paret hitta bättre lösningar vad gäller mjukvarukvalitet och underhåll jämfört med två individuella programmerare. Detta var särskilt framträdande när programmeringsproblemet var nytt för paret och krävde en ökad ansträngning för att utveckla design, algoritmer och koda programmet.

Princip två är att parprogrammering kan minska avsevärt i produktivitet när ett par har tidigare erfarenhet av samma uppgift och paret ännu inte har glömt den erfarenheten. Denna princip tar inte upp någon förändring av mjukvarans kvalitet. Det står helt enkelt det faktum att soloprogrammering är snabbare än parprogrammering när programmerare arbetar med lösningar de redan har stött på. Att ha en välutvecklad programmeringslösning är en fördelaktig situation, då det gör det möjligt för den som använder tangentbordet och musen att fortsätta skriva utan avbrott. Även om små stavfel kan

förekomma, anses det vara effektivt att inte avbryta skrivprocessen. Å andra sidan känner sig hans partner förmodligen mindre utmanad av att titta på den kända lösningen.

Lui och Chan (2006) drog även slutsatsen med hjälp utav deras två principer att nybörjare-nybörjare-par, jämfört med ensamarbetande nybörjare, är mer produktiva när det kommer till förfluten tid och mjukvarukvalitet. Dessutom fann de att expert-expert-par inte var lika effektiva som nybörjare-nybörjare-par när det gäller dessa faktorer jämfört med ensamarbetande experter.

2.3 Kunskapsöverföring

Enligt Plonka et al. (2015) sker det alltid en viss grad av kunskapsöverföring i en session av parprogrammering. Deras forskning visade också att dessa sessioner mellan experter och nybörjare inte bara fokuserar på kunskapsöverföring från experten till nybörjaren. De erbjuder också lärandemöjligheter för experter genom nybörjarens annorlunda perspektiv på problemet och befintlig kod. Genom att ställa enkla frågor kan nybörjaren utmana expertens befintliga idéer och bidra till att identifiera och lösa eventuella problem i koden. Dessutom kan nybörjaren föreslå lösningar som experten inte tidigare har övervägt. Nybörjare har ett *nybörjarsinne*, vilket innebär att de som saknar erfarenhet kan överväga fler möjligheter än en expert. Det påpekas även att nybörjare kan ställa frågor som hjälper till att avslöja experternas antaganden. Ett annat potentiellt sätt för experter att lära sig är genom verbalisering. Genom att prata om ett problem kan de få en bättre förståelse för själva problemet.

Rollen de agerar som i parprogrammering spelar också en roll. När experter agerar som förare tenderar de att engagera nybörjaren genom att förklara och verbalisera. När de agerar som navigatör ger de instruktioner och vägleder nybörjaren. Nybörjarens beteende påverkas också av om de är förare eller navigatör. När nybörjaren är förare uppmuntras de att tänka igenom, förstå och lösa problem på egen hand. Det är värt att notera att både experter och nybörjare var överens om att det är fördelaktigt för nybörjare att vara förare. Dock tar det längre tid att göra framsteg när nybörjaren är förare jämfört med när experten är förare (Plonka et al., 2015).

2.4 Mobbprogrammering

Ståhl och Mårtensson (2021) anser att mobbprogrammering är en framväxande praxis inom industrin som har fått ökad uppmärksamhet av forskare. De betonar att trots att mobbprogrammering kan verka som ett enkelt koncept, liknande parprogrammering men med fler personer. Det väcker liknande frågor som parprogrammering, fast i större utsträckning. De ifrågasätter effektiviteten i mobbprogrammering och om det passar alla utvecklare.

Vissa intervjupersoner hävdar att mobbprogrammering kan öka produktiviteten på lång sikt, men det saknas kvantitativt stöd för denna uppfattning. Andra betonar optimering av ledtid och presenterar trovärdiga argument, även här saknas det kvantitativt stöd. Det föreslås även att mobbprogrammering lämpar sig bäst för specifika typer av uppgifter, medan andra bör utföras individuellt. Det är svårt att se någon fördel med att ha flera utvecklare som tittar på när en person utför en enkel uppgift där det saknas en diskussion. En av de största fördelarna med mobbprogrammering verkar vara förmågan att uppnå tillräckligt med kompetens inom ett problemområde. Detta kan bidra till minskade ledtider. En nackdel med detta kan vara att deltagare som inte bidrar med något, kan vara överflödiga. (Ståhl & Mårtensson, 2021)

3 Metod

3.1 Litteraturstudie

En litteraturstudie består normalt av två delar. I den första delen utforskar studenter litteraturen för att hitta en lämplig forskningsidé och upptäcka relevant material om eventuella forskningsämnen. Detta underlättar studenterna att få en uppfattning om området och definiera ett forskningsproblem. Den andra delen av litteraturstudien påbörjas när ett ämne har valts. Den pågår sedan under hela forskningstiden, inklusive skrivandet av avhandlingen eller uppsatsen. Målet är att samla och presentera bevis för att stödja påståendet att man har skapat ny kunskap. (Oates, 2006)

Litteraturstudien genomfördes genom att söka efter litteratur i Google Scholar, som är Googles verktyg för att söka akademiska texter, samt Summon som är en söktjänst för att hitta Högskolan Dalarnas biblioteks tryckta och elektroniska material, de flesta fulltextartiklar samt vissa databaser.

Sökord som har använts i denna studie:

- Pair programming
- Mob programming
- Pair programming in software development
- Pair programming benefits
- Solo programming

3.2 Strategi: Fallstudie

I en fallstudie fokuserar man på en specifik instans av det fenomen som ska undersökas, till exempel en organisation, en avdelning, ett informationssystem, ett diskussionsforum, en systemutvecklare, ett projekt eller ett beslut. Denna enskilda instans, eller fall, studeras ingående med hjälp av olika metoder för datainsamling: intervjuer, observation, dokumentanalys, och enkäter. Målet är att få en djup och detaljerad inblick i fallets “liv” och dess komplexa relationer och processer. (Oates, 2006)

En fallstudie genomfördes på den statliga myndigheten Trafikverket. Syftet med fallstudien var att undersöka hur par-, mobb- och soloprogrammering används inom myndigheten och vilka skillnader som utvecklarna upplever med dessa metoder. En enkät användes för att samla in data från ett större antal utvecklare inom myndigheten och för att utvärdera utvecklarnas upplevelse och åsikter om par-, mobb- och soloprogrammering. Intervjuer användes för att få en mer kvalitativ data. Sammanfattningsvis bidrog båda datainsamlingsmetoderna till att ge en bredare och mer heltäckande förståelse för användningen av par-, mobb- och soloprogrammering inom Trafikverket. Vi skickade ut enkäten till 205 utvecklare där 23 stycken svarade, samt så intervjuade vi fyra utvecklare som inte deltog i enkätundersökningen för att undvika redundant data.

Datainsamlingen har bestått av en kombination av intervjuer och en enkätstudie och därmed har metodtriangulering använts. Metodtriangulering är användningen av mer än en datainsamlingsmetod för att bekräfta resultat och öka deras validitet (Oates, 2006). Denna mångsidiga data ger en djupare förståelse av ämnet och möjliggör en omfattande analys av de olika aspekterna av par-, mobb- och soloprogrammering som har undersökts.

3.3 Datainsamling

3.3.1 Intervjuer

En intervju är en speciell typ av konversation mellan människor. Vanligtvis har en person ett syfte med intervjun: de vill få information från den/de andra. Det innebär att diskussionen inte sker av en slump, utan har planerats av forskaren. De har vanligtvis en agenda, särskilda frågor de vill ha svar på, så diskussionens ämnen uppstår inte slumpmässigt. Forskaren styr diskussionen mot de ämnen som är av intresse för dem. I semistrukturerade intervjuer har forskaren en lista med teman som ska täckas och

frågor som de vill ställa. De är redo att ändra ordningen på frågorna beroende på samtalets utveckling. De kan även ställa ytterligare frågor om intervjupersonen tar upp ämnen som forskaren inte hade förberett frågor för. Intervjupersonen har möjlighet att ge mer detaljerad information om de ämnen som tas upp. Intervjupersonen kan också introducera ämnen som de anser vara relevanta för forskarens tema. (Oates, 2006)

Datainsamlingen för denna undersökning består av två delar: intervjuer med fyra utvecklare på Trafikverket samt en enkätstudie genomförd på Trafikverket. Dessa två delar av datainsamlingen ger en omfattande och mångsidig mängd data som möjliggör en grundlig analys av ämnet.

De fyra semistrukturerade intervjuerna med utvecklare på Trafikverket har bidragit med värdefull empiriska data. Dessa intervjuer gav möjlighet att få insikt i hur utvecklare på Trafikverket arbetar, vilka erfarenheter de har av par-, mobb- och soloprogrammering och vilka utmaningar de har stött på vid användning av dessa metoder. Intervjuerna gav också möjlighet att undersöka hur utvecklare på Trafikverket ser på effektiviteten av de olika programmeringsmetoderna. Urvalet gjordes utav vår handledare på Trafikverket som valde ut personer som har arbetat med parprogrammering på Trafikverket.

Frågorna konstruerades genom att identifiera teman i relevant litteratur och använda dem som grund för frågeformuleringen. Målet var att fånga utvecklarnas subjektiva upplevelser av par-, mobb- och soloprogrammering på ett välgrundat sätt.

3.3.2 Enkät

En enkät är en fördefinierad uppsättning frågor samlade i en förutbestämd ordning. Respondenter ombeds svara på frågorna och på så sätt tillhandahålla forskaren data som kan analyseras och tolkas. Enkäter är ofta förknippade men forskningsstrategin surveyundersökning. Där en enkät skickas ut till ett urval personer som ombeds fylla i den. Forskaren analyserar sedan alla svar, letar efter mönster och gör generaliseringar om handlingar och åsikter hos en större population än urvalet. Enkäter kan dock också användas inom andra forskningsstrategier, som en fallstudie. (Oates, 2006)

Enkätstudien genomförd på Trafikverket har bidragit med kvantitativa data och ett större urval av respondenter. Enkäten gav möjlighet att samla in data om hur vanligt förekommande par-, mobb- och soloprogrammering är på Trafikverket. Hur effektiva dessa metoder anses vara av utvecklarna och vilka fördelar och nackdelar som identifierats med användningen av dem. Enkäten distribuerades genom att skicka ut en uppmaning om deltagande i enkäten i en mailgrupp för kompetensforum inom systemutveckling på Trafikverket. Totalt skickades 205 enkäter ut och vi fick 23 svar, vilket motsvarar en svarsfrekvens på 11,2%. Enkäten var tillgänglig för svar under en period av sex dagar. Endast respondenter som svarade att de hade erfarenhet av parprogrammering och/eller mobbprogrammering gavs möjlighet att svara på frågor som berörde dessa områden. Det har inte kontrollerats på något sätt om respondenterna faktiskt har erfarenhet av par- och mobbprogrammering.

3.4 Etiska överväganden

För att vara en etisk forskare bör du behandla alla som är involverade i din forskning, oavsett om det är direkt eller indirekt, på ett rättvist och ärligt sätt. Deltagarna i forskningen har rättigheter som bör beaktas. När det gäller deltagande i forskning har individer och företag rätt att välja att inte delta om de så önskar. Som forskare bör man respektera deras beslut och inte försöka tvinga dem genom övertalning eller hot. Även om någon samtycker till att delta har de rätt att ändra sig när som helst och välja att dra tillbaka sitt deltagande. Individer har rätt till att ge informerat samtycke. Det innebär att om de samtycker till att delta, ges deras samtycke endast när de först har fått fullständig information om forskningens karaktär och deras medverkan. Deltagare i forskningen har rätt till anonymitet och skydd av identitet och plats. Deltagare i forskningen har rätt till sekretess när det gäller de data som samlas in från dem. Det innebär att forskaren har ansvaret att skydda deltagarnas identitet och säkerställa att data hålls konfidentiella. (Oates, 2006)

Som deltagare i denna studie tillkommer följande rättigheter:

- Rätten att välja att inte delta i studien utan motivering. Deltagande är frivilligt och det finns inga negativa konsekvenser för att avstå från att medverka.
- Rätten att när som helst avbryta deltagandet i studien utan krav på förklaring. Detta kan göras genom att meddela forskningsansvarig eller studiens samordnare.
- Rätten att få tydlig och begriplig information om studiens syfte, metod, förväntade resultat samt eventuella risker eller fördelar. Tillräcklig tid och möjlighet ska ges för att ställa frågor och fatta ett välgrundat beslut om deltagande baserat på tillhandahållen information.
- Garanterad konfidentialitet av din identitet och personlig information. Insamlade data kommer att anonymiseras och presenteras på ett sätt som inte möjliggör koppling till individuell deltagare. Dina svar och bidrag kommer inte att associeras med din personliga identitet i forskningsresultat eller eventuell publicering.
- All information som samlas in under studien kommer att behandlas konfidentiellt och endast vara tillgänglig för forskningsteamet.

3.5 Analysmetod

3.5.1 Kvalitativ analys

Kvalitativa data omfattar alla icke-numeriska data – ord, bilder, ljud och så vidare – som finns i intervjuinspelningar, forskares dagböcker, företagsdokument, webbplatser och utvecklares modeller. Det utgör den huvudsakliga typen av data eller evidens som genereras av fallstudier. I de flesta fall av kvalitativ dataanalys innebär det att abstrahera från forskningsdata, de verbala, visuella eller auditiva teman och mönster som anses vara viktiga för forskningsämnet. (Oates, 2006)

De kvalitativa data sammanställdes genom att vi spelade in intervjuerna. Sedan användes Microsoft Words transkriberingsfunktion för att generera en transkribering som vi sedan gick tillbaka och rätta till där verktyget gjort fel. Det fanns även fåtal kvalitativa frågor i enkäten. Genom analys av det kvalitativa datamaterialet framkom flera teman relaterad till styrkor och svagheter som utvecklare upplever vid användning av par-, mobb- och soloprogrammering.

- Kodkvalitet
- Kunskapsöverföring
- Tidsåtgång
- Skillnader mellan nyutveckling och förvaltning

3.5.2 Kvantitativ analys

Kvantitativa data innebär numeriskdata eller evidens. Det är den primära formen av data som genereras genom experiment och enkäter. Tanken med dataanalys är att söka efter mönster i datan och dra slutsatser. Det finns ett brett spektrum av etablerade tekniker för att analysera kvantitativa data. En enkel analys skulle använda tabeller, diagram eller grafer. Dessa gör det möjligt för forskaren eller läsaren att se mönster. (Oates, 2006)

Vi har valt att göra en enkät där vi tar in intervalldata, ordinaldata, datakodning samt kvalitativdata. Där vi i datakodningen använt oss av både fördefinierade frågor men även alternativ för egna svar samt öppna frågor med helt egna svar. Vi har valt att använda oss av tabeller då dessa är lämpliga för alla typer av data. (Oates, 2006)

4 Empiri

4.1 Deltagarnas bakgrund

Respondenternas arbetslivserfarenhet som utvecklare visade sig vara varierad. En liten andel hade mindre än ett års erfarenhet, medan andra hade mellan 1-3 års erfarenhet. Ingen hade arbetat mellan 4-6 år. En större andel hade en arbetslivserfarenhet på 7-10 år, medan majoriteten hade över 10 års erfarenhet som utvecklare. Det fanns även intervjupersoner med över 10 års erfarenhet, en med 7-10 års erfarenhet och en med 1-3 års erfarenhet.

När det gällde erfarenheten av parprogrammering, var den också varierad bland respondenterna. Vissa hade ingen tidigare erfarenhet, medan andra hade mindre än 1 års erfarenhet. Det fanns även de som hade mellan 1-2 års erfarenhet och några med 3-4 års erfarenhet av parprogrammering. En betydande andel hade över 5 års erfarenhet av parprogrammering. Alla intervjupersoner hade någon form av erfarenhet av parprogrammering, även om det inte nämndes hur omfattande denna erfarenhet var.

Vad gäller mobbprogrammering var det likaså en varierad bild bland respondenterna. En del hade ingen tidigare erfarenhet, medan andra hade mindre än 1 års erfarenhet. Det fanns även de med 1-2 års erfarenhet, men ingen hade 3-4 års erfarenhet. Endast en liten andel hade över 5 års erfarenhet av mobbprogrammering. Alla intervjupersoner hade någon form av erfarenhet av mobbprogrammering, även om det inte nämndes hur omfattande denna erfarenhet var.

I den fjärde delen framkom det att några respondenterna för närvarande parprogrammerade, medan en betydande andel inte gjorde det. Majoriteten parprogrammerade ibland, medan en mindre andel hade tidigare erfarenhet av parprogrammering.

Vidare visade det sig att respondenternas sätt att parprogrammera var varierat. En del satt fysiskt tillsammans och parprogrammerade, medan andra arbetade på distans. Många följde en blandning av båda metoderna.

Slutligen framkom det att respondenternas uppfattning om att parprogrammera på distans kontra på plats var varierad. Vissa såg ingen skillnad, medan en andel ansåg att det fungerade bättre att sitta på plats. Det fanns också de som tyckte att det var enklare att arbeta på distans och de som fann det svårare. Dessutom förekom 11% ogiltiga svar.

4.2 Par- och mobbprogrammerings styrkor

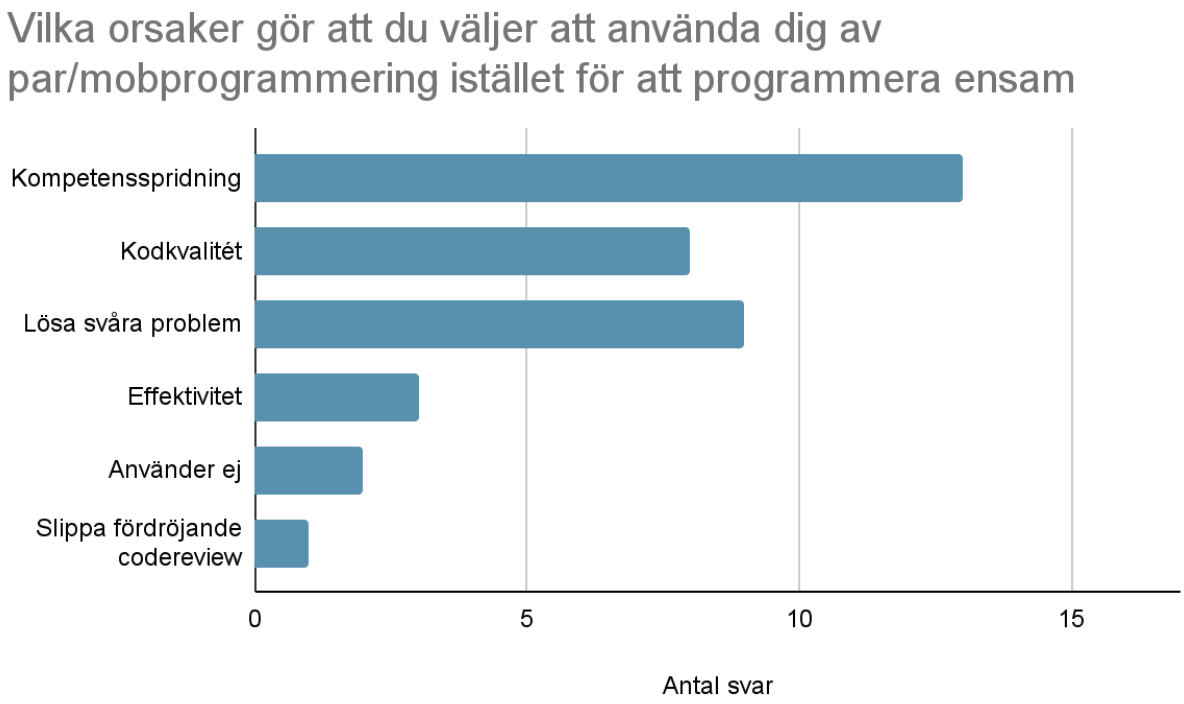


Fig 1. Visar varför utvecklare väljer par/mobbprogrammering istället för att programmera ensam

I figur 1 kan vi se att respondenterna hade olika motiv för att välja par- eller mobbprogrammering istället för att arbeta ensamma. En grupp uppgav att de gjorde det för att dela och sprida kunskap (kompetensspridning), medan andra fokuserade på att förbättra kodkvaliteten. En tredje grupp valde denna metod för att lösa svåra problem. Det fanns också en mindre grupp som kombinerade kompetensspridning och kodkvalitet som skäl för sitt val.

Intressant nog valde ingen att enbart använda par- eller mobbprogrammering för att öka effektiviteten. En grupp valde det för att både sprida kompetens och förbättra kodkvaliteten. Vissa respondenter använde inte par- eller mobbprogrammering alls.

En större grupp av respondenterna valde att använda detta tillvägagångssätt för både kompetensspridning och för att lösa svåra problem. En mindre grupp inkluderade även målet att undvika tidskrävande kodgranskningar som en anledning.

Det är tydligt att de flesta respondenterna identifierade kompetensspridning som den primära drivkraften bakom deras val att använda par- eller mobbprogrammering.

4.3 Par- och mobbprogrammerings svagheter

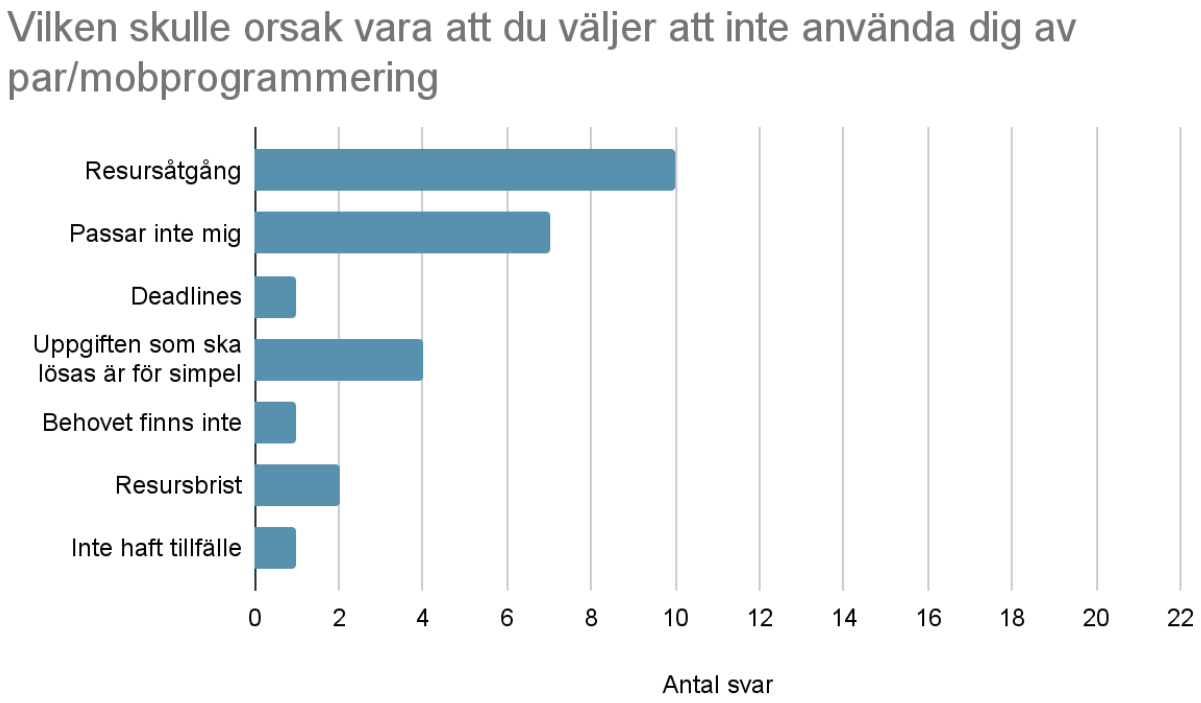


Fig 2. Illustrerar de olika motiv som utvecklare har för att avstå från att använda sig av par- eller mobbprogrammering.

De vanligaste motiven för att avstå från par- eller mobbprogrammering inkluderar bland annat en hög andel av respondenterna som svarade att det beror på resursåtgången. Denna orsak utgjorde en betydande faktor för många utvecklare. För en del utvecklare handlade det om att samarbetssättet helt enkelt inte passade deras arbetsstil eller personlighet, vilket utgjorde den näst vanligaste orsaken.

Andra svarade att de inte hade haft tillfälle att prova på par- eller mobbprogrammering, medan vissa ansåg att det var svårt att förena resursåtgång med uppsatta deadlines. En mindre andel av respondenterna ansåg att samarbetssättet inte var fördelaktigt när det gällde att lösa uppgifter som involverade mainstreamkod, och en del ansåg att de föredrog att arbeta själva.

Slutligen nämnde några utvecklare att de inte hade tillräckligt med kollegor att samarbeta med, medan andra kände sig kapabla att lösa uppgifter på egen hand. Dessa olika motiv ger insikt i varför vissa utvecklare avstår från att använda par- eller mobbprogrammering och visar på mångfalden av perspektiv och arbetspreferenser inom utvecklingsvärlden.

4.4 Par-, mobb- och soloprogrammering i nyutveckling

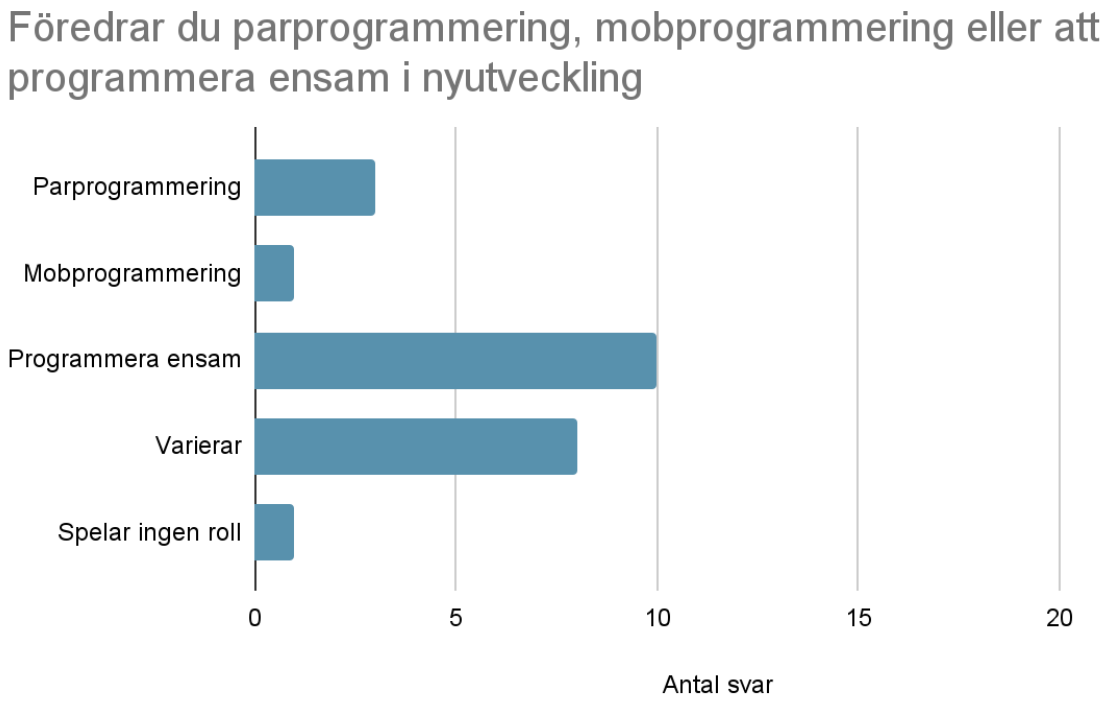


Fig 3. Visar utvecklarnas preferenser i nyutveckling

I nyutveckling så föredrar en majoritet av utvecklarna att programmera ensam eller så tycker de att det varierar. Det är en väldigt liten del av utvecklarna som föredrar mobbprogrammering samt tycker att det inte spelar någon roll. Det var en något större del men fortfarande ganska liten som föredrar att parprogrammera i nyutveckling.

De utvecklare som föredrar parprogrammering i nyutveckling framhåller främst fördelarna med att arbetet går snabbare och att antalet fel och buggar minskar, vilket leder till en bättre lösning.

En liten del av utvecklarna föredrog mobbprogrammering i nyutveckling. Fördelarna som utvecklarna såg med mobbprogrammering är att man får en större kompetensspridning.

Utvecklare föredrar att programmera ensam i nyutveckling, primärt på grund av begränsade resurser för par- eller mobbprogrammering. De är vana med soloprogrammering och upplever det som mer effektivt.

Utvecklarna upplevde att det kan variera vad de föredrog. Par- och mobbprogrammering är mer ansträngande och kräver mer pauser. Par- och mobbprogrammering är mest gynnsamma i komplexa uppgifter där deras samarbetsfördelar utmärker sig. I mindre komplexa projekt kan de upplevas som överflödiga och mindre produktiva. Har en utvecklare mindre erfarenhet är det bra med parprogrammering. Mindre grupper har inte riktigt tid för att kunna använda par- eller mobbprogrammering. Nya funktioner och innovationer i nyutveckling är bra att mobbprogrammera då hela gruppen får en förståelse för hur projektet ska fungera medans den enklare koden är effektivast att skriva ensam. Par- och mobbprogrammering fungerar bättre vid större uppgifter.

4.5 Par-, mobb- och soloprogrammering i förvaltning

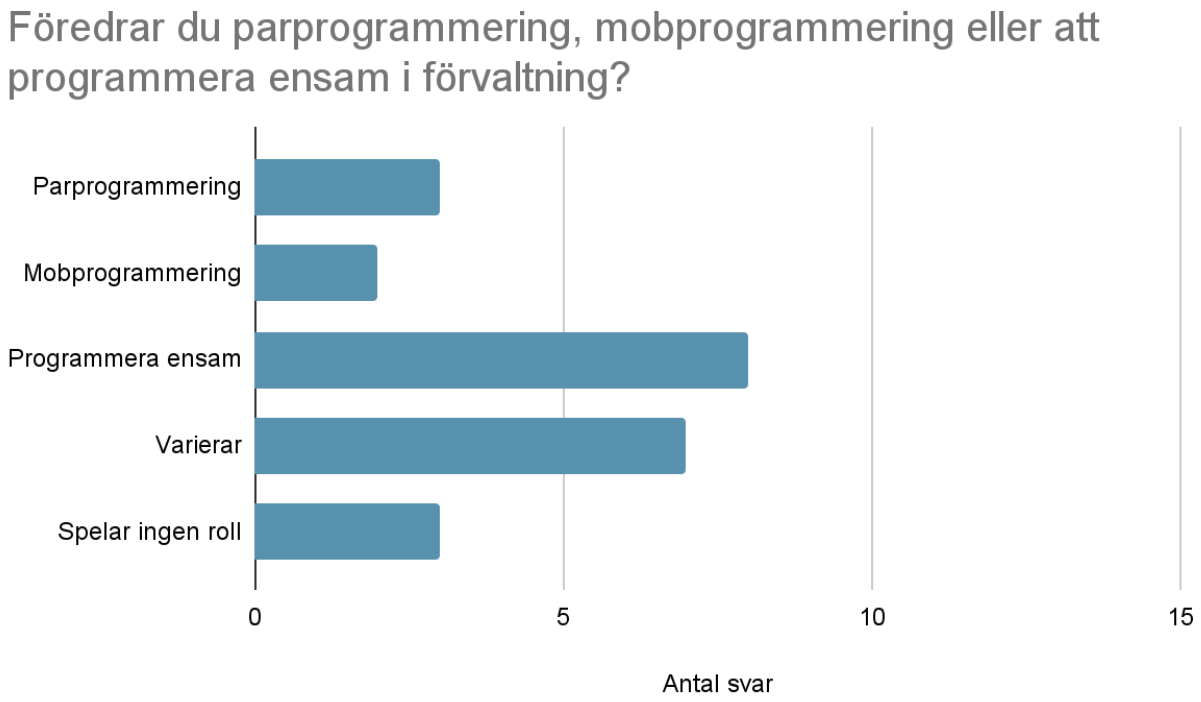


Fig 4. Visar utvecklarnas preferenser i förvaltning

De som föredrar att använda parprogrammering inom förvaltning framhåller flera fördelar med denna metod. För det första, det ger dem möjlighet att bättre lära känna varandra och underlättar samtidigt kodgranskning. Dessutom betraktas det som mer effektivt och mindre problematiskt än mobbprogrammering, som kan bli komplicerad med för många deltagare. En ytterligare fördel som lyfts fram är att parprogrammering minskar risken för att förändringar oavsiktligt påverkar befintlig funktionalitet som kanske hade blivit bortglömd annars.

I diskussionen om fördelarna med mobbprogrammering i förvaltning framkom olika synpunkter på varför några föredrar detta arbetssätt. En av de åsikter som lyftes fram var att det är mer givande och roligare att arbeta på detta sätt jämfört med andra metoder. En annan respondent betonade att mobbprogrammering erbjuder en effektiv och bra möjlighet till samarbete, vilket anses vara en av dess styrkor.

I diskussionen om fördelarna med ensam programmering i förvaltning framkom olika synpunkter på varför vissa föredrar detta arbetssätt. En respondent betonade att de känner att de lär sig bäst när de arbetar ensamma och att samarbete kan göra dem mer passiva samt förlora insikten i koden. En annan respondent menade att ensam programmering är en inrotad vana som de trivs med och anser vara mer effektivt och snabbare än att arbeta i grupp. Vidare nämndes brist på erfarenhet som en möjlig orsak till att föredra ensamarbete. Slutligen uttryckte en annan respondent att de uppskattar friheten att arbeta på sitt eget sätt och att de kan koncentrera sig bättre när de arbetar i tystnad.

I sammanhanget kring variationen i arbetssätt inom förvaltningen framkom en mängd olika faktorer som påverkar varför olika tillvägagångssätt används. Dessa aspekter inkluderade:

- Vid vissa tillfällen ansågs det vara nödvändigt att involvera flera personer för att förstå och granska komplex kod noggrant.
- Parprogrammering identifierades som en effektiv metod för att sprida kunskap och för att genomföra avancerad felsökning. Samtidigt ansågs den vara resurskrävande och mindre lämplig för mindre buggfixar och enklare förbättringar.

- Variationen i arbetssätt relaterades också till kodens svårighetsgrad. I de fall där koden var särskilt komplicerad eller om förståelsen för koden var osäker, ansågs det vara motiverat att ha ytterligare en person involverad i arbetet.
- En annan faktor som diskuterades var att par- eller mobbprogrammering ibland kunde vara mer ansträngande och kräva fler pauser än ensamarbete.
- Det framkom även att olika uppgifter och situationer kräver olika arbetssätt. Ibland kan det vara fördelaktigt att arbeta ensam för att analysera och lösa problem i lugn och ro.

Efter att deltagarna fått sätta en skala på hur de upplevde de olika arbetssätten, fick vi följande kommentarer om hur arbetssätten fungerade i olika situationer:
En person nämnde att det beror på om utvecklarna är bekväma med par- eller mobbprogrammering och att erfarna utvecklare oftast har arbetat ensamma hela karriären. En annan person påpekade att det generellt sett fungerar mest effektivt att programmera ensam, men att det kan vara användbart att få in flera ögon i vissa situationer, särskilt när man är osäker på bästa tillvägagångssätt. Det nämndes också att när det är många personer kan det vara svårt att komma överens och att det kräver en bra teamkänsla och prestigelöshet för att lyckas oavsett arbetssätt.

Slutligen nämndes att vissa personer är svåra att övertyga om att arbeta tillsammans om de föredrar att programmera ensamma. En person påpekade också att verktyget Skype inte var bra för samarbetet och att det saknas andra alternativ. En respondent kommenterade att de hade för lite erfarenhet av par- eller mobbprogrammering för att kunna säga, men att kommunikation är nyckeln oavsett vilket arbetssätt man väljer.

Enligt svaren kan vi se de största skillnaderna mellan att arbeta i nyutveckling och förvaltning när det gäller par-, mobb- och soloprogrammering. När det kommer till förvaltning framhävs vikten av att förstå äldre kod, där många ögon kan vara till hjälp. Andra utvecklare som har jobbat längre med systemen kan ha en bättre översikt och bidra till att förstå koden. mobbprogrammering i förvaltning nämns som ett bra sätt att sprida kompetens. I nyutveckling är det istället viktigt att hitta bra lösningar tillsammans som ett team.

Tidsaspekten skiljer sig också, då nyutveckling ofta har mindre tid till förfogande. En fördel med att arbeta tillsammans, antingen i par eller som en mobb, är att det leder till bättre kod som är lättare att förstå och mer korrekt. I nyutveckling kan det vara utmanande att par- eller mobbprogrammera eftersom det finns många kreativa idéer om design och arkitektur. I förvaltning är fokus istället på att lösa uppgiften snarare än att komma med innovativa lösningar.

När man programmerar ensam i nyutveckling finns risken att utvecklaren sätter sitt personliga prägel på koden, vilket kan leda till att olika delar av samma projekt har olika design. Rutinerna och reglerna är inte etablerade i nyutveckling, medan förvaltning oftast har ett fungerande system med befintliga ramar och rutiner, vilket underlättar arbetet med innehållet.

En respondent påpekar att det inte finns stora skillnader mellan arbetssätten och att det kan variera beroende på kända problemområden eller system i förvaltningen. För nyutveckling krävs mer tid för att förstå och skapa något från grunden, medan förvaltning handlar om att vidareutveckla och förbättra något som redan finns.

Sammanfattningsvis visar svaren att att dela idéer och samarbeta med kollegor kan vara fördelaktigt oavsett arbetssätt. Flexibilitet och möjligheten att anpassa arbetssättet efter uppgiften framhävs som viktiga faktorer.

4.6 Par-, mobb- och soloprogrammering i nyutveckling och förvaltning

Tabell 1
Utvecklarna sätter en skala på 1-5 hur bra eller dåligt dem upplever att par-, mobb- och soloprogrammering fungerar i nyutveckling och förvaltning.

Hur upplever du att de olika sätten fungerar i de olika situationerna?							
		Dåligt		Okej		Mycket bra	
Parprogrammering i nyutveckling?	(%)	1 9	2 4	3 13	4 43	5 22	Ej Tillämpligt 9
Mobbprogrammering i nyutveckling?	(%)	1 13	2 9	3 4	4 26	5 17	Ej tillämpligt 30
Programmera ensam i nyutveckling?	(%)	1 0	2 4	3 26	4 26	5 39	Ej tillämpligt 4
Parprogrammering i förvaltning?	(%)	1 4	2 4	3 9	4 52	5 22	Ej tillämpligt 9
Mobbprogrammering i förvaltning?	(%)	1 9	2 9	3 9	4 26	5 17	Ej tillämpligt 30
Programmera ensam i förvaltning?	(%)	1 0	2 4	3 30	4 26	5 35	Ej tillämpligt 4

Resultaten tyder på att i både nyutveckling och förvaltning är parprogrammering överväldigande positivt bedömt. Dock verkar det vara en starkare positiv inställning till parprogrammering i förvaltning än i nyutveckling.

De allra flesta tycker att mobbprogrammering fungerar bra till mycket bra i både nyutveckling och förvaltning, medan en del anser att det inte är tillämpligt. Det finns också en minoritet som anser att det fungerar dåligt eller är mellan dåligt och okej.

Resultaten tyder på att soloprogrammering i både nyutveckling och förvaltning är övervägande positivt. Majoriteten anser att det fungerar bra eller mycket bra, medan en del tycker att det är okej. Det är också noterbart att det är väldigt få med negativ syn på soloprogrammering i både nyutveckling och förvaltning.

4.7 Intervjuer

4.7.1 Intervju 1

Intervjupersonen har en omfattande erfarenhet av att vara utvecklare sedan 1997 och arbetar inom förvaltning men uttrycker att vissa delar av nyutveckling sker inom förvaltning.Intervjupersonen visar

sig vara mer bekanta med parprogrammering än mobbprogrammering, vilket intervjupersonen har engagerat sig i veckovis respektive endast ett fåtal gånger.

Intervjupersonen framhåller en preferens för soloprogrammering, parprogrammering nämns också som en metod som används ibland, medan mobbprogrammering är mer sällsynt för intervjupersonen. Vid parprogrammering förekommer både fysisk närvaro och distansarbete. Intervjupersonen anser att båda tillvägagångssätten, oavsett om det utförs på plats eller på distans, är effektiva. Dock föredrar intervjupersonen personligen att vara fysiskt närvarande på samma plats för att möjliggöra direkt interaktion och användning av whiteboard.

Gällande skillnaderna mellan par-, mobb- och soloprogrammering noterar intervjupersonen att soloprogrammering möjliggör mer självständighet och snabbare framsteg. Men med risk för att fastna på vissa uppgifter. Parprogrammering, även om det är långsammare ger stabilitet och underlättar kompetensspridning. mobbprogrammering, som är mindre vanligt i intervjupersonens erfarenhet erbjuder ännu större kompetensspridning för att det är fler deltagare men framstegen upplevs ännu långsammare.

När det kommer till kommunikation påpekar intervjupersonen att parprogrammering uppmuntrar direkt och omedelbar interaktion mellan deltagarna. Medan mobbprogrammering kräver mer eftertanke för att undvika att prata över varandra. Skillnaderna i effektivitet och tidsåtgång mellan par-, mobb- och soloprogrammering nämns också. Där intervjupersonen spekulerar att soloprogrammering kan vara snabbaste vad gäller omedelbara framsteg, men att parprogrammering med tanke på faktorer som kodkvalitet och kompetensspridning kan vara lika eller mer effektivt.

Intervjupersonen föreslår att både par- och mobbprogrammering resulterar i högre kodkvalitet på grund av de olika perspektiven och ökade läromöjligheter för deltagarna. Intervjupersonen betonar att soloprogrammering kan ha lägre kvalitet jämfört med par- och mobbprogrammering. Intervjupersonen Jobbar med “pullrequest” kodgranskningar så att de får lite av de aspekterna, även om en utvecklare soloprogrammerar.

När det kommer till skillnader i erfarenhet och kunskap mellan utvecklare påpekar intervjupersonen att både mindre erfarna och mer erfarna utvecklare har positiva aspekter nyttjar båda. Den mindre erfarna utvecklaren kan tillföra nya idéer och ha en fräsch syn på problemen, medan en mer erfaren utvecklare kan bidra med värdefull kunskap om verksamheten. Om erfarenhetsnivån är relativt likvärdiga kan det uppstå ett mer direkt utbyte av kunskap, medan olikheterna kan leda till positiva resultat.

Vidare framkom det att intervjupersonen föredrar en kombination av par- och soloprogrammering, där intervjupersonen vanligtvis arbetar självständigt men samarbetar i par när det är nödvändigt. Anledningen till att parprogrammering bara används ibland är för att det upplevs som för intensivt att jobba i par i långa perioder. Intervjupersonen har haft dålig erfarenhet av mobbprogrammering tidigare men tycker att det fortfarande fyller ett syfte.

När det gäller situationer som lämpar sig för parprogrammering framhåller intervjupersonen att det är fördelaktigt för problemlösning och kompetensspridning. Intervjupersonen anser att genom att diskutera och utmana varandras tankar kan positiva resultat uppnås.

Å andra sidan visade det sig att soloprogrammering lämpade sig bättre för rutinmässiga och mindre utmanande uppgifter. Där det inte fanns en tydlig fördel med att arbeta i par. Det noterades att soloprogrammering kunde vara mer effektivt och tidsbesparande i sådana situationer.

För nyutveckling och förvaltning identifierar intervjupersonen olika styrkor med parprogrammering. I nyutveckling ser intervjupersonen en större fördel med parprogrammering, särskilt vid etablering av nya grunder och identifiering av framåtriktade strategier. Medan förvaltning kan upplevas som enklare för soloprogrammerare om man inte stöter på större problem.

När det gäller svagheter med parprogrammering nämner intervjupersonen att det kan innebära onödig tidsåtgång inom både förvaltning och nyutveckling på kort sikt, men kan vara fördelaktigt på lång sikt.

Intervjupersonen anser att kompetensspridningen är en betydande fördel med mobbprogrammering. Det noterades också att intervjupersonen inte hade erfarenhet av att mobbprogrammera inom nyutveckling, utan snarare i samband med förvaltningsuppgifter. Det framkom att kompetensspridning var ett mål inom förvaltningsprojektet, där syftet var att dela och överföra kunskap till andra medlemmar i gruppen.

När det kommer till svagheter med mobbprogrammering så är det likt parprogrammering, men tidsåtgången upplevs vara längre i mob då det är fler deltagare. Det framkom även att vissa deltagare kanske inte känner sig helt bekväma eller vågar ta initiativ när de kommer in i en ny grupp. I detta avseende ansågs parprogrammering vara fördelaktigt, eftersom det skapar en mindre och tryggare miljö där deltagare vågar bidra och uttrycka sina idéer. Dessutom betonades vikten av att inte överge enskilda utvecklare utan istället sträva efter att skapa avstämningsmöjligheter och perioder av parprogrammering för att underlätta inlärning och utveckling.

Slutligen framkom det att intervjupersonen inte upplevde en stor skillnad mellan nyutveckling och förvaltning. Det ansågs att även förvaltningsprojekt innebar nya problem att lösa och utveckling av nya funktioner. Därför upplevdes inte övergången mellan dessa två områden som betydande.

4.7.2 Intervju 2

Intervjupersonen som för närvarande arbetar som systemvetare och har ungefär 2 års erfarenhet. Större delen av arbetet består av förvaltningsuppgifter, medan nyutveckling endast utgör en mindre del.

Intervjupersonen berättade att de har erfarenhet av parprogrammering och lite mobbprogrammering, och att det har varit ovanligt att mobbprogrammera. Intervjupersonen har spenderat mycket tid åt att parprogrammera med en senior utvecklare, vilket resulterat i kunskapsöverföring mellan dem. De har också varit flexibla med arbetssättet och har möjlighet att både sitta tillsammans fysiskt och arbeta på distans. Intervjupersonen upplever inte någon större skillnad i hur arbetet fungerar beroende på om de arbetar på distans eller på plats. Intervjupersonen noterar dock att när de parprogrammerar tillsammans på kontoret, kan det finnas fler distraktioner och sociala interaktioner att ta hänsyn till.

Enligt intervjupersonen är de största skillnaderna mellan att parprogrammera och programmera ensam det utbytet av erfarenheter och kunskaper som sker. Att arbeta tillsammans med en mer erfaren utvecklare ger dem möjligheten att lära sig och förstå olika metoder och koncept. Det finns också en fördel med att ha någon att diskutera och resonera med under arbetets gång.

När det gäller kodkvalitet anser intervjupersonen att parprogrammering har en positiv inverkan. Genom att ha flera ögon och hjärnor som granskar koden minskar risken för att misstag eller brister förbises. Intervjupersonen nämner att det är enklare att lära sig och förstå genom att jobba tillsammans jämfört med att förlita sig på dokumentation eller individuella tillvägagångssätt.

Intervjupersonen påpekar att det finns både fördelar och nackdelar med att arbeta med utvecklare med olika erfarenhetsnivåer och kunskaper. Att säkerställa att alla är med och förstår kan ibland kräva mer tid och anpassning. Men det är värt det eftersom det ger oss möjlighet att dra nytta av olika perspektiv och kunskaper. När alla delar med sig av sin kunskap ökar vår förståelse och berikar våra diskussioner och samarbeten.

Slutligen uttrycker intervjupersonen en preferens för parprogrammering, då intervjupersonen tycker att det är roligt och gillar att arbeta med andra människor. Intervjupersonen anser att det ger en bättre arbetsmiljö och möjligheten att bidra och lära sig från varandra.

4.7.3 Intervju 3

Intervjupersonen har arbetat som utvecklare sedan 1988 och för närvarande arbetar inom förvaltning. Intervjupersonen har erfarenhet av att programmera ensam, i par och i mobb, där majoriteten av deras arbete utförs ensam (95% ensam, 5% i par och mobb). Parprogrammering har använts tidigare under introduktion till ett nytt projekt.

När det gäller parprogrammering har de använt både fysisk närvaro och distansarbete. Intervjupersonen anser att det är bättre att arbeta i samma lokal på grund av kommunikation och användningen av Whiteboard. Intervjupersonen ser den största skillnaden mellan att programmera ensam och par/mobb som ansvaret. Arbetar man ensam så har man fullt ansvar medan par och mobb innebär att man kan luta sig på andras expertis.

Kommunikation i parprogrammering upplevs vara mer beroende av den mest insatte, medan ensamarbete kräver att man själv är väl insatt. När det gäller effektivitet och tidsåtgång anses parprogrammering vara mer effektivt vid kunskapsspridning och vid att dra nytta av olika kompetenser. Dock kan idéer ibland bli överröstade i gruppen.

När det gäller kodkvalitet anser Intervjupersonen att det inte finns någon betydande skillnad mellan parprogrammering och att programmera ensam. Parprogrammering kan dock leda till att kodstilen likriktas. Vidare betonar intervjupersonen vikten av att inkludera alla i par- och mobbprogrammering och att undvika att de som är blyga eller mindre erfarna inte hörs. De föreslår att man använder tydliga regler för att rotera mellan rollerna (förare & navigatör) och att diskutera syftet med par/mobbprogrammering för att undvika missförstånd.

När det gäller lämplighet för parprogrammering anser intervjupersonen att kunskapsspridning och svåra uppgifter kan dra nytta av det. Mobbprogrammering anses vara liknande parprogrammering, och utvecklaren ser ingen tydlig skillnad mellan dem. Intervjupersonen påpekar dock att det är viktigt att inte ha för stora team vid mobbprogrammering.

Slutligen betonar intervjupersonen vikten av att anpassa sättet att programmera efter individuella preferenser och att prata om det i teamet. Intervjupersonen anser att olika team kan ha olika preferenser och att det är viktigt att alla trivs och är med på samma sätt att programmera.

4.7.4 Intervju 4

Intervjupersonen har arbetat som utvecklare i 8 år och har erfarenhet av både parprogrammering och mobbprogrammering. När det gäller mobbprogrammering har intervjupersonen begränsad erfarenhet, och har gjort det ungefär 3 gånger. När det kommer till parprogrammering uppskattar intervjupersonen att 60% till 70% av tiden ägnas till parprogrammering. Intervjupersonen anser att de för närvarande ägnar mer tid åt att vägleda och lära ut i sitt arbete än att aktivt parprogrammera.

När intervjupersonen har parprogrammerat har det skett fysiskt, och det finns ingen erfarenhet av att parprogrammera på distans. Intervjupersonen hävdar att parprogrammering leder till kunskapsspridning och att tankeprocessen blir utmanad genom konstant input från en partner. Kommunikationen har inte upplevts annorlunda mellan par- och mobbprogrammering, eftersom intervjupersonen vanligtvis har arbetat med samma personer i båda situationerna.

När det gäller effektivitet och tidsåtgång så anser intervjupersonen att mobbprogrammering tar längre tid, men det anses vara mer effektivt för kunskapsspridning. Det bidrar även till högre kodkvalitet med hjälp av flera utvecklares input. Intervjupersonen noterar att soloprogrammering kan vara snabbare på att lösa specifika uppgifter. Däremot betonar intervjupersonen att par- och mobbprogrammering har fördelar när det gäller kunskapsspridning och underlätta samarbetet.

Intervjupersonen föredrar att parprogrammera eller mobbprogrammera när det finns behov av att lära sig något eller sprida kunskap. När det gäller uppgifter som de är bekanta med och känner sig säkra på, så föredrar intervjupersonen att arbeta ensamma. Intervjupersonen anser att genom att arbeta på egen hand kan de vara mer tidseffektiva och uppnå resultat snabbare. Detta beror på att de inte behöver spendera tid på att samordna eller dela kunskap med andra utvecklare.

När det kommer till svårigheter kan tidspress och olika kunskapsnivåer vara utmaningar i både nyutvecklings- och förvaltningskontexter. För att minimera dessa brister, föreslår Intervjupersonen att det är viktigt att etablera en tydlig och detaljerad kommunikation om kunskapsnivåer och förmågor från början. Detta underlättar korrekta tidsuppskattningar och effektiv planering. Genom att klargöra och

tydligt kommunicera med varandra om teammedlemmarnas kompetens skapas en grund för att förbättra arbetsflödet.

Mobbprogrammering främjar kunskapsspridning genom att sprida kunskap till alla teammedlemmar och undvika att förlita sig på en enda utvecklare. Det kan dock vara svårt att hantera tidsbegränsningar.

5 Analys

5.1 Styrkor och svagheter med soloprogrammering inom nyutveckling och förvaltning

Styrkor med soloprogrammering:

- Tidsåtgång: Soloprogrammering anses vara den mest tidseffektiva metoden för att lösa specifika uppgifter, eftersom man inte behöver samordna eller dela kunskap med andra utvecklare.
- Självständighet: Genom att arbeta ensam kan utvecklare ta fullt ansvar och ha kontroll över hela utvecklingsprocessen utan att förlita sig på andra.
- Flexibilitet: Soloprogrammering ger utrymme för utvecklaren att följa sina egna preferenser och arbetssätt utan att behöva anpassa sig till andra utvecklares arbetsstilar och rutiner.

Svagheter med soloprogrammering:

- Ingen kunskapsspridning: Genom att arbeta ensam kan det vara svårare att dela och sprida kunskap. Det kan leda till en mer begränsad kunskapsbas och mindre utbyte av idéer och perspektiv.
- Kodkvalitet: Utan att ha någon som granskar koden när den skrivs kan det leda till brister i kodkvalitet. Det kan även göra det svårare att upptäcka och korrigera potentiella fel i koden.

Det går att se att de flesta deltagare föredrar att programmera ensam. Detta framgår i fig 3 där 43% föredrar att programmera ensam i nyutveckling. Samt 35% som föredrar att programmera ensam i förvaltning vilket kan ses i fig 4. Man kan även se att 39% tycker att programmera ensam fungerar mycket bra i nyutveckling samt 35% som tycker att programmera ensam fungerar mycket bra i förvaltning. Detta kan ses i fig 5.

Soloprogrammering ses som lämpligast vid enklare uppgifter där utvecklaren redan har kompetens, och att det inte finns någon fördel med att sprida kompetens. Utvecklarna som deltog i denna studie hade varierande åsikter kring skillnaderna mellan nyutveckling och förvaltning. Vissa tyckte att par- och mobbprogrammering var lämpligare i nyutveckling, medan andra tyckte att det var lämpligare i förvaltning. Det fanns de som tyckte att det bästa tillvägagångssättet varierar beroende på sammanhanget, vilket gäller både inom nyutveckling och förvaltning.

Detta behöver dock inte betyda att programmera ensam fungerar bättre. De som föredrar att programmera ensam besvarade varför. Då framkom svar som “Gammal vana”, “Mest att jag inte har någon större erfarenhet av par- eller mobbprogrammering” och “resursbrist” vilket går att se i bilaga 1 fråga 12. Det går även att se liknande svar såsom “Gammal vana”, “dåligt med erfarenhet” och “får göra som jag vill, tänka själv, koncentrera mig i tystnad, göra mitt” vilket går att se i bilaga 1 fråga 15.

5.2 Styrkor och svagheter med parprogrammering inom nyutveckling och förvaltning

Styrkor med parprogrammering:

- Kunskapsspridning: Parprogrammering främjar spridningen av kunskap och kompetens inom teamet. Genom att arbeta tillsammans kan utvecklarna dela och utbyta sina idéer och expertis. Det leder till ökad kunskap hos utvecklarna och i teamet som helhet.
- Förbättrad problemlösning: Genom att kombinera olika perspektiv och erfarenheter kan parprogrammering underlätta problemlösning. Två utvecklare kan samarbeta och utmana varandras idéer för att hitta de bästa lösningarna.
- Högre kodkvalitét: Det finns en till person som granskar koden och kan komma med feedback. Samt utbyta idéer och perspektiv

Svagheter med parprogrammering:

- Tidsåtgång: Det anses att det tar längre tid att göra framsteg i parprogrammering jämfört med soloprogrammering.

- Ökad resursanvändning: Parprogrammering kräver att två personer arbetar tillsammans på en och samma uppgift. Detta innebär att det krävs dubbelt så många resurser jämfört med soloprogrammering.
- Passar inte alla: Parprogrammering kräver att två utvecklare arbetar tillsammans och samarbetar. Det kan uppstå svårigheter om utvecklarna har olika arbetsstilar, personligheter eller kommunikationsmönster.

De största styrkorna med parprogrammering är främst kunskapsspridningen och att lösa svårare problem. Det kan vara mer tidseffektivt att arbeta i par för att lösa vissa problem än att lösa det ensam. Kodkvaliteten upplevs även vara högre när den utvecklas i par. Parprogrammering kan vara lika eller mer tidseffektivt som soloprogrammering om man tar hänsyn till faktorer som kunskapsspridning och kodkvalitet.

5.3 Styrkor och svagheter med mobbprogrammering inom nyutveckling och förvaltning

Styrkor med mobbprogrammering:

- Kunskapsspridning: Möjligheten för kunskapsspridning är som störst i mobbprogrammering eftersom det är fler deltagare jämfört med par- och soloprogrammering.
- Lösa svåra problem enklare: Det är enklare att lösa problem när det är flera utvecklare som kan komma med förslag och idéer på lösningar.
- Högre kodkvalitet: Det är flera personer som granskar koden samtidigt som den skrivs. Detta kan leda till högre kodkvalitet.

Svagheter med mobbprogrammering:

- Tidsåtgång: Det anses att det tar längst tid att göra framsteg jämfört med par- och soloprogrammering.
- Resursbrist: Det krävs ännu fler resurser än parprogrammering eftersom det krävs minst tre personer för att kallas mobbprogrammering.
- Passar inte alla: mobbprogrammering passar inte alla arbetsstilar och personligheter.

Den största styrkan med mobbprogrammering är främst kunskapsspridningen. Det är värt att notera att majoriteten av utvecklarna i denna studie hade begränsad erfarenhet av mobbprogrammering. Många utvecklare betraktade även parprogrammering och mobbprogrammering som liknande, med skillnaden främst i antalet deltagare.

6 Diskussion och slutsatser

6.1 Diskussion

Baserat på de presenterade styrkorna och svagheterna med soloprogrammering, parprogrammering och mobbprogrammering kan vi dra några slutsatser angående valet av tillvägagångssätt inom både nyutvecklings- och förvaltningskontexter.

Soloprogrammering framstår som den mest tidseffektiva metoden för att lösa specifika uppgifter, eftersom utvecklaren inte behöver samordna eller dela kunskap med andra. Det ger också självständighet och kontroll över hela utvecklingsprocessen. Å andra sidan kan soloprogrammering leda till en begränsad kunskapsbas och brister i kodkvalitet, då det saknas kunskapsspridning och eventuell granskning av koden.

Parprogrammering främjar kunskapsspridning och kompetensutbyte inom teamet. Genom att arbeta tillsammans kan utvecklarna dela idéer och erfarenheter, vilket kan leda till högre kodkvalitet och förbättrad problemlösning. Nackdelarna med parprogrammering inkluderar längre tidsåtgång och ökad resursanvändning, eftersom två personer måste arbeta tillsammans på samma uppgift. Dessutom kan det uppstå svårigheter om utvecklarna har olika arbetsstilar eller personligheter.

Mobbprogrammering erbjuder störst potential för kunskapsspridning, då fler deltagare är inblandade. Det kan leda till en högre kodkvalitet och underlätta lösningen av svåra problem genom att flera utvecklare kan bidra med förslag och idéer. Dock kan mobbprogrammering kräva mer resurser och tid än de andra tillvägagångssätten.

6.2 Slutsatser

Det är viktigt att välja rätt tillvägagångssätt beroende på de specifika målen och kraven för projektet. Soloprogrammering kan vara fördelaktigt när snabbhet och självständighet är avgörande, men det kan leda till begränsad kunskapsspridning och brister i kodkvalitet. Eftersom många av deltagarna i studien hade omfattande erfarenhet som utvecklare, kan vi observera likheter med resultatet som Lui och Chan (2006) fann. I huvudsak tyder resultaten på att parprogrammering blir mindre effektiv när utvecklaren redan har erfarenhet av liknande uppgifter.

Parprogrammering kan främja kunskapsdelning och förbättra kodkvaliteten, men det kan kräva mer tid och resurser samt kan vara utmanande om utvecklarna har olikheter i arbetsstil eller kommunikation. Kunskapsspridning och lösa svåra problem är de främsta anledningarna till att utvecklare valde parprogrammering. Det går att se likheter till vad Lui och Chan (2006) kom fram till i deras första princip där parprogrammering blir mycket effektivare när programmeringsproblemet är nytt för utvecklarna vilket det ofta kan vara om det anses vara ett svårare problem.

Mobbprogrammering erbjuder störst potential för kunskapsspridning och hög kodkvalitet, men det är också den mest resurskrävande metoden. Studien visar att erfarenheten av mobbprogrammering är väldigt liten jämfört med parprogrammering och i de fall man använt mobbprogrammering är just för att sprida kunskap till ännu fler utvecklare.

För att välja rätt tillvägagångssätt bör man noggrant överväga projektets krav, teamets kompetenser och dynamiken mellan utvecklarna. Genom att göra en välinformerad bedömning kan man öka chanserna till framgång och maximera effektiviteten i både nyutveckling och förvaltning, samtidigt som man utnyttjar resurserna på bästa möjliga sätt.

6.3 Vidare forskning

Framtida forskning inom mjukvaruutveckling kan fokusera på att objektivt jämföra effektivitet, produktivitet och kodkvalitet för par-, mobb- och soloprogrammering när det finns olika erfarenhet hos utvecklarna. Studier bör också undersöka hur olika tillvägagångssätt påverkar kunskapsdelning, inlärning, arbetsmiljö och samarbete inom utvecklingsteam. Genom att få en bättre förståelse för dessa faktorer kan forskningen ge vägledning för att välja det mest lämpliga tillvägagångssättet baserat på specifika projekt och behov i teamet.

7 Referenser | References

Agile Application Development. (2023). How do you measure the quality and productivity of pair programming vs solo programming? https://www.linkedin.com/advice/1/how-do-you-measure-quality-productivity

CERISE-projektet. (2014). Parprogrammering. Hämtad från https://www.csc.kth.se/tcs/projects/cerise/parprogrammering/ (Senast uppdaterad 2020).

Buchan, J., & Pearl, M. (2018). *Leveraging the Mob Mentality: An Experience Report on Mob Programming*. https://doi.org/10.1145/3210459.3210482

Lui, K. M., & Chan, K. C. C. (2006). Pair programming productivity: Novice–novice vs. expert–expert. International Journal of Human-Computer Studies, 64(9), 915-925. https://doi.org/10.1016/j.ijhcs.2006.04.010

Oates, B. J. D. (2005;2012;2006;). *Researching information systems and computing*. SAGE Publications Ltd.

Plonka, L., Sharp, H., Van Der Linden, J., & Dittrich, Y. (2015). Knowledge transfer in pair programming: An in-depth analysis. *International Journal of Human-computer Studies, 73*, 66–78. https://doi.org/10.1016/j.ijhcs.2014.09.001

Ståhl, D., & Martensson, T. (2021). Mob programming: From avant-garde experimentation to established practice. *Journal of Systems and Software, 180,* 111017. https://doi.org/10.1016/j.jss.2021.111017

Sun, W., Marakas, G., & Aguirre-Urreta, M. (2016). The effectiveness of pair programming: Software professionals' perceptions. *IEEE Software, 33*(4), 72-79. https://doi.org/10.1109/MS.2015.106

Trafikverket. (2023) Om oss - Vår verksamhet, vision och uppdrag. Hämtad från https://www.trafikverket.se/om-oss/var-verksamhet-vision-och-uppdrag/

Williams, L., & Kessler, R. (2000). All I really need to know about pair programming I learned in kindergarten. Communications of the ACM, 43(5), 108-114. https://doi.org/10.1145/332833.332848

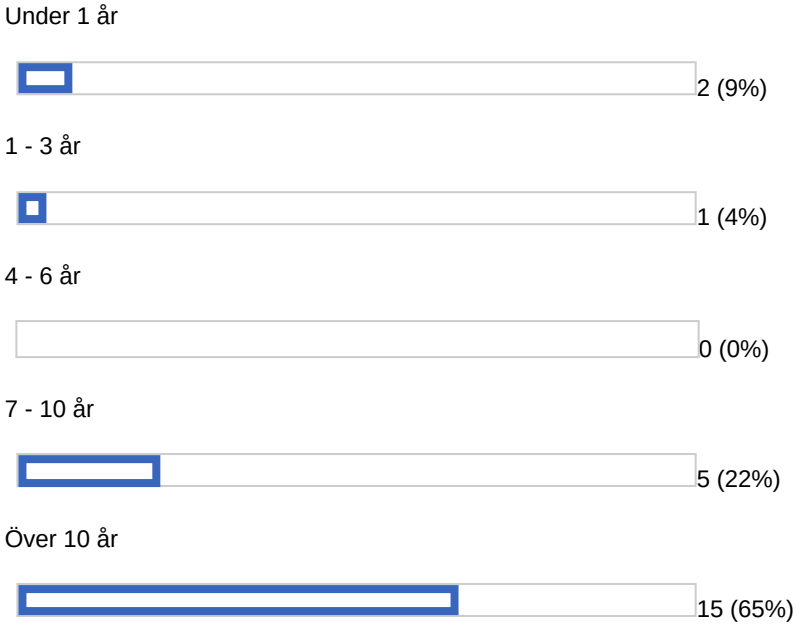
Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. IEEE Software, 17(4), 19–25. https://ieeexplore.ieee.org/abstract/document/854064

Zuill, W., & Meadows, K. (2016). Mob programming: A whole team approach. In *Agile 2014 Conference, Orlando, Florida* (Vol. 3).

8 Bilagor

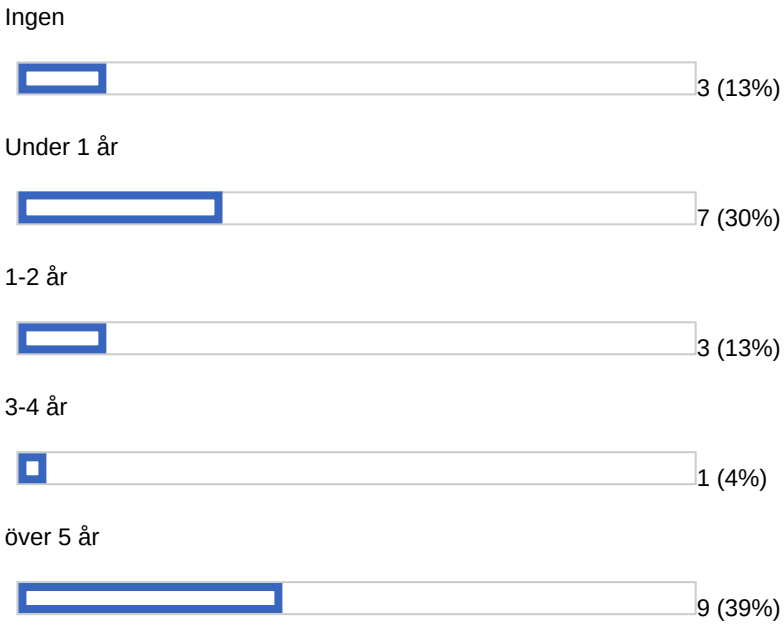
8.1 Bilaga 1 Enkätundersökning

9 Hur många år har du arbetat som utvecklare?



Summa: 23

10 Hur lång erfarenhet har du kring parprogrammering?



Summa: 23

11 Hur lång erfarenhet har du kring mobbprogrammering (Programmera i grupp)?

Ingen

12 (52%)

Under 1 år

7 (30%)

1-2 år

3 (13%)

3-4 år

0 (0%)

över 5 år

1 (4%)

Sum

12 Parprogrammerar du i dagsläget?

Ja

1 (4%)

Nej

8 (35%)

Ibland

12 (52%)

Nej men har gjort förut

2 (9%)

Summa: 23

13 Sitter ni vid samma dator eller gör ni det på distans via skärmdelning t.ex.?

Samma dator

2 (13%)

Distans

3 (20%)

Både och

10 (67%)

Summa: 15

14 Upplever du några skillnader mellan sätten att parprogrammera på?

På distans kan den som inte skriver kod, söka eller göra annat parallellt

1 (11%)

Poäng med par programmering är att lära varandra. Görs det på samma eller olika datorer spelar ingen roll.

1 (11%)

Nej, ingen större skillnad.

1 (11%)

Nej

1 (11%)

Det är enklare att sitta vid samma dator än på distans. Dels kan du peka på skärmen och diskutera, samt rita på papper bredvid och dessutom så ser man på den skriver koden när hen inte hänger med. Den som programmerar har alltid nackdel genom den personen behöva skriva koden, komma med ideér och samtidigt lyssna på den andra personen.

1 (11%)

Bättre fokus när man sitter tillsammans.

1 (11%) 

Nej.

1 (11%) 

Vid parprogrammering är det bara två som behöver komma överens. Vid mobbprogrammering tror jag att det är svårare att komma fram till ett gemensamt beslut över HUR man ska göra.

1 (11%) 

Det är skitkrångligt att parprogrammera via Skype

1 (11%) 

Summa: 9

10. Varför föredrar du parprogrammering i nyutveckling?

man får mer gjort snabbare, och vi är just nu för få för att bilda mer än par

Det minskar riska att man glömmer/missar att ta höjd för ovanliga men möjliga kombinationer/scenarier, därmed minskar och risken för buggar. När man tillsammans diskuterar koden samtidigt som den skrivs kommer man många gånger fram till bättre lösningar än om man sitter själv.

Summa: 2

11. Varför föredrar du mobbprogrammering i nyutveckling?

Man får bättre kompetenstäckning och fångar fler problemställningar tidigt som man kan ta höjd för i lösningen.

Summa: 1

12. Varför föredrar du att programmera ensam i nyutveckling?

Känner mig friare att prova mig fram till en hållbar lösning

Gammal vana

Jag tycker att det är mer effektivt.

får göra som jag vill, tänka själv, koncentrera mig i tystnad, göra mitt jobb.

Parprogrammering kräver att man är på ungefär samma nivå för att det inte ska bli att man "lär ut" eller blir "lärd" istället för att man faktiskt jobbar effektivt med att lösa ett problem.

Det går allt som oftast snabbast.

Mest att jag inte har någon större erfarenhet av par- eller mobb-
programmering.

Resursbrist

fanns inte alternativet "det enda jag har provat" jag har sällan möjlighet att parprogrammera, men den erfarenhet jag har av det kommer från när jag eller någon annan behöver extra ögon på ett problem.

Summa: 9

13. Vad är orsaken till att det varierar i nyutveckling?

Det är mer ansträngande att jobba med par/mobb-programmering. Kräver att man tar mer
pauser.

Mer komplexa grejer kan vara bra att ett helt team är med på. När det gäller mindre komplexa delar men där en person har markant mindre erfarenhet är det bra med parprogrammering. Men det mesta går bra att göra på egen hand, det är också då man lär sig som bäst när man är nybörjare - enligt mig.

Det beror på teams storlek. Mindre team hinner inte. Stora team kan passa
bättre.

T.ex. boilerplatekod är oftast mycket klipp och klistra från tidigare projekt, här kan det vara fördel att mobbprogrammera för att sprida kunskap, men är det ett team som redan kan samma sak så går det fortast sätta upp projektet genom att dela upp uppgifter mellan varandra. Ny funktionalitet och innovativa lösningar i nyutveckling vinner på att mobbprogrammera då får hela får hela teamet förståelse för hur det nya projektet ska fungera. I alla projekt förekommer det och bara mainstreamkodning, d.v.s. kod som bara ska skrivas där de kluriga delarna redan har lösts. Dessa delar är effektivast att koda ensam

Vissa uppgifter är så tydliga/enkla att det är effektivast att en utför det. Andra uppgifter kan vara komplexa både ur domän och IT-perspektiv och då kompletterar ett par varann och båda lär sig mer, dvs win-win :)

Det beror på förutsättningar och sammanhang. Om utvecklingsresurser exempelvis har markant olika förkunskaper/kompetens inom området kan det vara väldigt fördelaktigt att par- eller mobbprogrammera, för att dela/sprida kompetens. I andra fall kan det vara ett område där man hellre sitter på egen hand och i lugn och ro analyserar ett problemområde eller löser en uppgift.

Beroende på uppdragets storlek och utseende så passar de olika utvecklingsstilarna olika bra. Par- och grupp-programmering passar bättre vid lite större projekt.

Det beror på vilket problem som ska lösas

Summa: 8

15. Varför föredrar du parprogrammering i förvaltning?

Då har man tid att lära varandra. Kodgranska på rätt sätt osv.

effektivare och för få/lite svårt med mob

Man minskar risken för att de förändringar man gör förstör någon gammal funktionalitet som man kanske inte hade kommit ihåg/glömt att den finns. Man är två person som kan komma på "ja just ja, vi måste tänka på det här också".

Summa: 3

16. Varför föredrar du mobbprogrammering i förvaltning?

Det är roligare

Ett roligt och bra sätt att arbeta tillsammans

Summa: 2

17. Varför föredrar du att programmera ensam i förvaltning?

För att jag lär mig bäst då. Annars blir det lätt att andra tar alla beslut och man själv är passiv, även om man håller i tangentbordet så hänger man inte med i vad som händer i koden.

Gammal vana

Jag tycker att det är mer effektivt.

får göra som jag vill, tänka själv, koncentrera mig i tystnad, göra mitt jobb.

Parprogrammering kräver att man är på ungefär samma nivå för att det inte ska bli att man "lär ut" eller blir "lärd" istället för att man faktiskt jobbar effektivt med att lösa ett problem.

Det går snabbast.

Dåligt med erfarenhet.

Summa: 7

18. Vad är orsaken till att det varierar i förvaltning?

Ibland behövs flera ögon för att förstå viss komplex kod

Det är mer ansträngande att jobba med par/mobb-programmering. Kräver att man tar mer pauser.

Kunskapsspridning, avancerad felsökning och programmering av kritiska funktioner är ypperliga exempel på då parprogrammering är att föredra. Vid enklare buggar och mindre förbättringar så tar det för mycket resurser att parprogrammera.

Vissa uppgifter är så tydliga/enkla att det är effektivast att en utför det. Andra uppgifter kan vara komplexa både ur domän och IT-perspektiv och då kompletterar ett par varann och båda lär sig mer, dvs win-win :)

Det beror på förutsättningar och sammanhang. Om utvecklingsresurser exempelvis har markant olika förkunskaper/kompetens inom området kan det vara väldigt fördelaktigt att par- eller mobbprogrammera, för att dela/sprida kompetens. I andra fall kan det vara ett område där man hellre sitter på egen hand och i lugn och ro analyserar ett problemområde eller löser en uppgift.

Passar olika bra beroende på vad det är för programmeringsuppdrag. Vid större projekt kräver det ofta mer programmeringsresurser.

kan bero på hur svårt det är att sätta sig in i koden. om systemet är invecklat kan förståelse för koden i sig ligga till grund för en viss osäkerhet. då vill man nog ha en till person med.

Summa: 7

20. Har du några kommentarer kring hur arbetssätten fungerar i dem olika situationerna?

Det är mer frågan om utvecklarna är bekväma i att arbeta med par/mobb-programmering. Passar inte alla och de flesta erfarna utvecklarna idag har solo-utvecklat hela karriären

För lite erfarenhet

Parprogrammering och mobbprogrammering är två olika sätt att utföra jobbet med glädje och sprida kunskaper. Men de är inte krav. Det är svårt att övertala visa människor som vill programmera ensamma.

Det går generellt mest effektivt för mig att programmera ensam, men det kan också vara nyttigt att få in flera ögon i vissa lägen (till exempel när man känner sig lite osäker på bäst tillvägagångssätt att lösa ett problem).

jo fler människor, ju fler viljor, tid slösas, svårt komma överens. ingen kan stå bakom beslut eftersom ingen fick sin vilja igenom osv.

-

Bra teamkänsla, prestigelöshet, "open mind" och engagemang krävs för framgång oavsett arbetssätt!

Jag tycker inte att något av arbetssätten fungerar bättre eller sämre baserat på om det är förvaltning eller nyutveckling. Det beror helt på sammanhanget och vilka resurser som är inblandade. Alla varianter kan (i teorin) fungera alldeles utmärkt oavsett situation, därav mina 5:or som svar.

mobbprogrammering har jag ingen erfarenhet av. Parprogrammering fungerar bäst om det man ska lösa är komplext och har många parametrar man måste ta hänsyn till. Ensamprogrammering fungerar bra när det som ska lösas är mer enkelt och rakt på.

Har inte så stort yrkeserfarenhet av annat är ensamprogrammering. Parprogrammering och gruppprogrammering har jag till viss del gjort i yrkeshögskola.

Skype är ett uselt verktyg i sammanhanget! Och tyvärr så har vi ju inget annat... Jag har för lite erfarenhet av par/mob för att egentligen säga, men så länge man kan kommunicera så ser jag inget problem oavsett vilken väg man väljer.

Summa: 12

21. Vad upplever du är de största skillnaderna mellan att arbeta i nyutveckling och förvaltning när det gäller par-, mobb- och att programmera ensam?

I förvaltning är det ofta äldre kod som man ska förstå och då kan många ögon vara bra. Andra utvecklare kan ju ha jobbat längre i systemen och har då bättre översikt

Vid förvaltning kan mob vara bra för kompetensspridning. I nyutveckling är det mer att tillsammans hitta bra lösningar.

När det gäller nyutveckling är det dels bra att hela teamet har en förståelse för hur koden fungerar från början. Dels att vi är som starkast tillsammans, när alla får vara med och bidra i realtid, då är mobbprogrammering en bra taktik.

Inga synpunkter

Tid. Nyutveckling har ofta inte så mycket tid.

Har inte upplevt någon skillnad.

ingen.

det blir vettigare kod som är lättare att förstå och mer korrekt när man är flera som måste förstå det på en gång

Par-/mobprogrammera i nyutveckling kan vara utmanande då många är kreativa och det finns många ideér på hur man kan lösa design och arkitektur. I förvaltning är par/mobbprogrammering lugnare då fokus är på att lösa uppgiften snarare än att komma med kreativa och nytänkande coola lösningar.

Programmera ensam i nyutveckling kan vara en risk finns att utvecklare ser koden som "sin" och sätter mer sitt egna avtryck i koden, man kan se vem som skrivit vilken kod vilket gör att olika delar koden i samma projekt har olika design.

I nyutveckling finns inga rutiner, verksamhetsreglerna är inte satta osv. Detta gör att t ex parprogrammering funkar mycket bra här. I förvaltning har man i regel ett väl fungerade system med befintliga ramar och rutiner vilket gör det enkelt att jobba med innehållet.

Ingen större skillnad

Inga stora skillnader. Eventuellt kan det vara mer kända problemområden/system i förvaltning, vilket skulle kunna gynna par- och/eller mobbprogrammering, men det kan naturligtvis också variera.

Kommer inte på någon större skillnad. 1

Att jobba med nyutveckling betyder ju att inget finns från början, man måste lägga mer tid på att förstå vad det är som ska göras. När man jobbar i förvaltning handlar det om att ta hand om något som redan finns, vidareutveckla och förfina det.

Har, som sagt, inte så stor erfarenhet av de olika sätten att programmera här på Trafikverket eller i yrkeslivet. Men känner att jag skulle lära mig mer och snabbare om jag kan bolla idéer med en eller flera kollegor som också programmerar i samma eller liknande projekt.

Friare ramar ang val av lösning vid nyutveckling. För enkla buggfixar är det overkill att mobbprogrammera i förvaltning

jag har nästan ingen riktig erfarenhet av ren par--/mobbprogrammering, men sett till sammarbete överlag skulle jag säga att man under nyutveckling behöver vara duktiga på att dela en bild av vad lösningen kommer bli, så att alla går mot samma mål, medan förvaltning blir mer fokuserat på att hjälpas åt att få en tydlig bild av hur systemet är byggt och att veta hur vissa förändringar kommer påverka övriga delar av systemet.

Summa: 17

8.2 Bilaga 2 Intervjufrågor

1. Hur länge har du arbetat som utvecklare?
2. Jobbar du med en nyutveckling eller förvaltning för tillfället?
3. Har du erfarenhet av att jobba med parprogrammering & mobbprogrammering?
4. Kan du uppskatta hur mycket du har jobbat med solo, par och mob över åren?
5. Parprogrammerar du i dina nuvarande arbetsuppgifter?
6. Vad upplever du är de största skillnaderna mellan att programmera i par-, mobb- och solo?
7. Upplever du någon skillnad i kommunikation när du parprogrammerar eller mobprogrammerar?
8. Upplever du skillnader i effektivitet(tidsåtgång) när du programmerar i par-, mobb- och solo?
9. Upplever du skillnader i kodkvalitet i par-, mobb- och soloprogrammering?
10. Om det finns olika erfarenhet och kunskap mellan parterna (utvecklarna som samarbetar), hur påverkar det?
11. Föredrar du att parprogrammera, mobprogrammera eller programmera ensam (solo)?
12. Vilka styrkor upplever du att parprogrammering har i nyutveckling och förvaltning?
13. Vilka svagheter upplever du att parprogrammering har i nyutveckling och förvaltning?
14. Vilka styrkor upplever du att mobbprogrammering har i nyutveckling och förvaltning?
15. Vilka svagheter upplever du att mobbprogrammering har i nyutveckling och förvaltning?
16. Hur ser processen ut när ni blir tilldelade en parprogrammerings partner eller en mobbprogrammerings grupp?
17. Vilka typer av projekt eller uppgifter tror du är mest lämpliga för parprogrammering?
18. Vilka typer av projekt eller uppgifter tror du är mest lämpliga för mobbprogrammering?
19. Vilka typer av projekt eller uppgifter tror du är mest lämpliga för soloprogrammering?
20. Hur skiljer sig upplevelsen för en ny utvecklare som går med i ett team i ett nytt projekt jämfört med förvaltning (pågående projekt)?
21. Finns det något mer du vill tillägga när det kommer till nyutveckling och förvaltning?