



INSTITUTIONEN FÖR TILLÄMPAD IT

GENERATIVA AI-MODELLER INOM SYSTEMUTVECKLING

En explorativ studie över seniora systemutvecklarens
erfarenheter av Copilot

August Aublet

Gustaf Hasselgren

Mårten Jonsson

Kandidatuppsats:	15 hp
Ämne:	Informatik
År:	2023
Rapport nr:	2023:001

Sammanfattning

Det har länge diskuterats och utforskats gällande digitala hjälpmedel för systemutveckling. Idag finns det en mängd verktyg för att underlätta programmering. Artificiell intelligens spelar en allt större roll inom systemutveckling, och ett exempel på detta är verktyget Copilot. Copilot är ett kod-genererande verktyg som använder sig av en AI-modell (Codex) som tränats på kod genom Github-repositories. Tidigare studier visar på att Copilot kan göra systemutvecklare mer produktiva och minska frustration genom att ta över repetitiva uppgifter. Copilot har potential att revolutionera systemutvecklarens arbetssätt, men det finns begränsad forskning kring hur verktyget faktiskt används av erfarna professionella systemutvecklare i vardagen. Studiens syfte ämnar därför att undersöka och svara på frågan hur och varför seniora systemutvecklare använder sig av Copilot.

Studien genomfördes genom ett kvalitativt och explorativt angreppssätt. För insamling av data genomfördes tolv semistrukturerade intervjuer med professionella systemutvecklare med längre erfarenhet (fyra år eller mer). Utifrån analysen av detta material framkom sex teman som ämnar ge en rikare bild till att förstå forskningsfrågorna. Studiens resultat visar att respondenterna som har använt Copilot är nöjda med verktyget, men att de inte använt alla funktioner som verktyget har att erbjuda. Resultat tyder även på att deras användning av Copilot påverkas av tidigare arbetssätt, och de använder främst Copilot som ett autocomplete-verktyg. Det tyder även på att det finns potential för användare att kunna få ut mer nytta av verktyget genom utbildning. Att de valt att använda Copilot beror till stora delar på nyfikenhet till ny teknik och på att de får utrymme till att fokusera på mer meningsfulla uppgifter genom att Copilot effektiviserar kodskrivning. Det framkommer dock att verktyget har outnyttjad potential vilket i några fall går att komplettera med ChatGPT. Det finns således stora framtidsmöjligheter för AI-verktyg inom systemutveckling.

Nyckelord

Github Copilot, Generativa programmeringsmodeller, NLP-programmering, AI programmeringsmodeller, AI par-programmering

Förord

Vi vill rikta ett stort tack till samtliga respondenter som tagit sig tid att ställa upp i vår studie. Utan ert bidrag hade inte denna studie varit möjlig. Vi vill även rikta ett stort tack till Mattias Rost som har guidat oss med sina kunskaper och stöd.

Innehållsförteckning

1	Inledning.....	1
1.1	Syfte och frågeställning.....	2
2	Bakgrund.....	3
2.1	Copilot.....	3
2.2	Att använda Copilot.....	4
2.3	Copilots genererade kod.....	5
2.3.1	Copilots förmåga att lösa uppgifter.....	5
2.3.2	Genererad kod och läsbarhet.....	6
2.4	Parprogrammering.....	6
2.5	ChatGPT.....	7
3	Metod.....	9
3.1	Litteratursökning.....	9
3.2	Angreppssätt.....	9
3.3	Empiriskt urval.....	10
3.4	Datainsamling.....	12
3.5	Analys av data.....	13
4	Resultat.....	15
4.1	Respondenternas relation till programmering.....	15
4.1.1	Programmering en del i verktygslådan.....	15
4.1.2	Läsbar kod framför optimering.....	15
4.1.3	Tråkigt med repetitiva uppgifter.....	16
4.1.4	Flow i arbete.....	16
4.2	Varför började de använda Copilot?.....	17
4.3	Copilots användande.....	18
4.3.1	Hur används Copilot?.....	18
4.3.2	Copilots begränsningar.....	21
4.3.3	Reglering av Copilot.....	21
4.4	Nyttoeffekter av Copilot.....	22
4.4.1	Utbildande och utmanande.....	22

4.4.2	Minskad kognitiv ansträngning	23
4.4.3	Paradigmskifte inom programmering	23
4.5	ChatGPT, ett komplement till Copilot	24
4.6	Copilot som parprogrammerare	25
5	Diskussion	27
5.1	Effektivare arbete med Copilot	27
5.2	Promptar i naturligt språk: det utforskade arbetssättet.....	28
5.3	Komplex relation till större kod-stycken	29
5.4	”With great power comes”... samma ansvar som alltid	29
5.5	Parprogrammering: människa vs AI	30
5.6	Kvalitet på genererad kod	31
6	Reflektion	33
7	Slutsats.....	34
8	Referenser.....	35
9	Bilagor.....	40

1 Inledning

Digitala hjälpmedel vid systemutveckling har länge diskuterats och utforskats. Redan på 60-talet togs det fram automatiserade system för att underlätta och hjälpa människans logiska tänkande, och minska felmarginaler i manuellt arbete (Waldinger & Lee, 1969). Programmeringsspråk har stegvis gått från direkt maskinkod till högre nivåer av språk som i olika former abstraherar komplicerade implementationer (*Computer History*, u.å.). Idag finns verktyg för att underlätta programmeringsprocessen, så som Intellisense, vilket är ett verktyg som integreras i utvecklingsmiljön (IDE). Det kan ge förslag på funktions- och variabelnamn baserat på kod som redan är skriven i miljön. Intellisense är beskrivet som en form av *autocomplete*-verktyg, där det huvudsakliga syftet är att hjälpa till att komma ihåg namngivningar och minska knapptryck på tangentbordet (*IntelliSense in Visual Studio Code*, u.å.).

Intellisense har använts flitigt sedan det dök upp under 90-talet, men nu börjar nästa stora steg i utvecklingen. Artificiell intelligens (AI) får en allt större roll i samhället, och så även inom programmering (Welsh, 2022). Brown m.fl. (2020) beskriver att de senaste åren har stora genombrott i AI-utveckling lett till möjligheten att träna på enorma mängder data, och därmed att utveckla effektiva språkmodeller. En sådan modell är GPT-3, som fungerar på det sätt att den tar emot en input, prompt, som användaren skriver. Modellen försöker därefter att generera text, som enligt den data den har tränats med, borde följa efter prompten (Brown m.fl., 2020). Ur GPT-3 föddes en kod-inriktad generativ modell, Codex. Codex har tränats vidare på olika programmeringsspråk, mer specifikt på omkring 54 miljoner *repositories* på Github (*OpenAI Announces 12 Billion Parameter Code-Generation AI Codex*, u.å.). Codex i sig blev grunden till det kod-genererande verktyget Copilot. En betaversion av Copilot lanserades i oktober 2021, och i juni 2022 släpptes den färdiga kommersiella produkten (*GitHub Copilot · Your AI Pair Programmer*, u.å.). Initiala reaktioner kring Copilot samlades in i en enkät av det skapande företaget Github, och resultatet verkade mer än lovande (Kalliamvakou, 2022). Deltagarna berättade i enkäten att 88% upplevde sig mer produktiva, samtidigt som en majoritet kände mindre frustration, ett högre fokus och spenderade längre tid i ”*flow*”. Man upplevde också att man kunde lägga mer energi och tankekraft på det som är roligt, snarare än att behöva lägga tid på repetitiva uppgifter (Kalliamvakou, 2022).

Copilot marknadsförs av Github som ”din AI-par-programmerare”, vilket anspelar på ett etablerat arbetssätt inom systemutveckling (*GitHub Copilot · Your AI Pair Programmer*, u.å.). Parprogrammering går ut på att två systemutvecklare tillsammans sitter och jobbar med samma uppgift, där de har olika roller. En vanlig

fördelning är att en person ”kör”, alltså producerar kod, medan den andra personen ”navigerar”, alltså har en mer övergripande blick och väljer ut en riktning samtidigt som den granskar koden. Bird m.fl. (2022) beskriver att Copilots roll blir som förare, och systemutvecklaren blir navigeraren. Man kan då anta att vinsten i konceptet blir dels att Copilot möjliggör parprogrammering för alla systemutvecklare, även om man jobbar ensam, dels att arbetsgivare skulle kunna frigöra resurser som AI kan ersätta.

Det verkar som att Copilot har potential att underlätta, påverka och kanske även revolutionera systemutvecklarens arbetssätt. I och med att Copilot är ett så pass nytt verktyg så finns det dock begränsat med forskning. Mycket av forskningen som finns kretsar kring vad Copilot faktiskt klarar av att göra och hur effektivt resultatet blir (Denny m.fl., 2022; Finnie-Ansley m.fl., 2022, 2023). En del studier har genomförts kring användande som vill visa på effekten av Copilot (Barke m.fl., 2022; Bull & Kharrufa, 2023). Bara ett par studier har undersökt hur användare faktiskt upplever och använder Copilot, och de har genomförts i ett väldigt tidigt skede (Barke m.fl., 2022; Vaithilingam m.fl., 2022). Denna studie vill berika förståelsen kring verktyget Copilot genom att låta erfarna systemutvecklare berätta om sina erfarenheter med egna ord.

1.1 Syfte och frågeställning

Det växer ständigt fram nya metoder och verktyg för att effektivisera arbetssätt inom systemutveckling. Effektivisering handlar inte bara om att skriva snabbare kod utan berörs av många andra faktorer som samarbete, säkerhet och utforskande. I och med framstegen inom AI har man lyckats skapa verktyg som på en helt ny nivå kan interagera och förstå vardagligt språk men också träna på enormt stora mängder data vilket har öppnat upp för helt nya sorters hjälpmedel så som Copilot.

På grund av att Copilot lanserades så pass nyligen så är tillgången till forskning på området begränsad. Den forskning som finns har främst fokuserat på faktorer som användande och kodkvalitet, och har dessutom utförts i ett väldigt tidigt skede i Copilots livstid. Det finns alltså en avsaknad inom forskningen gällande det faktiska vardagliga användandet av Copilot i en professionell kontext och hur den personliga inställningen är utifrån seniora systemutvecklarens perspektiv.

Studiens syfte blir därför att undersöka seniora systemutvecklarens erfarenheter av att använda Copilot, för att skapa en rikare bild av Copilot som verktyg i en okontrollerad miljö. Utifrån detta så tar studien avstamp i följande definierad frågeställning:

Hur och varför använder seniora systemutvecklare Copilot?

2 Bakgrund

Detta avsnitt ämnar ge en djupare bild av den tidigare forskningen på området. Avsnittet börjar med att ge en beskrivning av Copilot och hur verktyget fungerar. Vidare redovisas hur tidigare studier valt att undersöka Copilot, och avsnittet avslutas med ett stycke om ett annat AI-verktyg, ChatGPT, som under studiens gång visade sig vara relevant och därmed togs med i bakgrunden.

2.1 Copilot

Wermelinger (2023) beskriver Copilot som ett hjälpmedel för programmering som integreras i IDE:n. Copilot försöker fylla på med förslag utifrån den kontext som ges och presenterar dessa när man tar en paus i skrivandet. Förslaget dyker upp i grått och kursivt typsnitt direkt efter där man skriver (se figur 1). Man har då valet att acceptera förslaget med knappen *Tab* eller avfärda med *Esc*. Detta arbetssätt kan kort beskrivas som *autocomplete* eller *code completion*. Man har även möjlighet att gå igenom flera alternativ av förslag. Trycker man *Ctrl+Enter* genererar Copilot tio förslag som man får upp i en panel vid sidan av där man sedan kan välja att acceptera det man tycker passar bäst (Wermelinger, 2023). Tillvägagångssättet har stora likheter med befintliga *autocomplete*-verktyg som är integrerade i IDE:n men skiljer sig åt genom att Copilot klarar av mycket större kontext samt har möjlighet att ge betydligt större kodförslag (Bird m.fl., 2022). Wermelinger (2023) beskriver också att Copilot även kan generera kod genom att man skriver promptar i naturligt språk (se figur 2). Det gör man genom att skriva kod-kommentarer, något som skiljer sig åt gentemot befintliga *autocomplete*-verktyg.

Figur 1

Copilot ger ett förslag (i grå kursiv text) till ett givet funktionsnamn.

```
1 def to_rövarspråket(text):
2     consonants = "bcdfghjklmnpqrstvwxyz"
3     new_text = ""
4     for letter in text:
5         if letter in consonants:
6             new_text += letter + "o" + letter
7         else:
8             new_text += letter
9     return new_text
```

Figur 2

Copilot ger ett förslag på en funktion efter en given prompt i naturligt språk (kommentaren högst upp i bilden).

```
1 #write a function that translates a text input into rövarspråket.
2
3 def rovarspraket(text):
4     consonants = "bcdfghjklmnpqrstvwxyz"
5     new_text = ""
6     for letter in text:
7         if letter in consonants:
8             new_text += letter + "o" + letter
9         else:
10            new_text += letter
11    return new_text
```

2.2 Att använda Copilot

Barke m.fl. (2022) har studerat användandet genom att låta testdeltagare lösa fyra bestämda uppgifter av något varierande sort, både enstaka uppgifter som ”*Advent of Code*” men även en chatt-klient och server för att få arbeta med kod i en något större kontext (*Advent of Code*, u.å.). Författarna delar i sin studie in användandet i två delar, *accelererande* och *undersökande*. *Accelererande* beskriver arbetet som sker när programmeraren redan vet hur hen skall lösa uppgiften. Då används Copilot primärt som en intelligent *autocomplete*. Man har stort fokus på att bryta ner uppgifterna till små komponenter för att på så vis lätt kunna verifiera förslagen. Man lägger också mindre tid på att rätta till ”fel” Copilot gör och väljer snarare att inte acceptera förslagen.

Barke m.fl. (2022) beskriver vidare att *undersökande* är när man försöker hitta nya lösningar eller lösa ett problem utanför eller i gränsen över sitt kunskapsområde. Man låter då Copilot ge större förslag som man sedan utvärderar och förfinar. Här kan kommentarer i naturligt språk underlätta för att beskriva problemet och ge Copilot kontext. Man tar även hjälp av Copilots multiförslags-funktion, som innebär att man ber Copilot generera tio förslag där man sen kan välja ett av dem för att implementera. Författarna beskriver även risk för ”*cognitive overload*”, alltså att det är svårt att ta till sig flera förslag. Man är även mer benägen att acceptera förslag som är nästan rätt för att sedan justera koden så att den utför det man vill (Barke m.fl., 2022).

Användandet av Copilot innebär också ett förändrat arbetsflöde där man i större grad granskar förslagen i stället för att skriva dem (Barke m.fl., 2022; Jiang m.fl., 2022; Mozannar m.fl., 2022). Barke m.fl. (2022) menar att deltagare i deras studie upplever att Copilot kan störa *flow*, när man hela tiden får förslag på kod och därmed måste ta ställning till om förslaget är vad man tänkte sig. För att kunna använda Copilot *accelererande* krävs det att man har en mycket god bild av hur man skall lösa uppgiften och kunna dela upp uppgiften i mindre delar (Barke m.fl., 2022). Att skriva *boilerplate*-kod, kod som är repetitiv, är något som flera studier påvisat att Copilot är bra på (Barke m.fl., 2022; Sarkar m.fl., 2022).

Denny m.fl. (2022) undersöker i en studie hur bra Copilot är på att generera kod utifrån promptar i naturligt språk. Författarna lyfter vikten av att arbeta med *prompt engineering*, d.v.s. att lära sig att skriva strukturerade prompts för att få ut så bra förslag som möjligt av Copilot. De lyfter två problemområden; dels att prompten kan vara för konceptuell, dels att prompten kan vara för lång (Denny m.fl., 2022).

2.3 Copilots genererade kod

2.3.1 Copilots förmåga att lösa uppgifter

Flera studier har forskat kring koden som Copilot kan generera (Dakhel m.fl., 2022; Finnie-Ansley m.fl., 2022, 2023). De har låtit Copilot få göra examineringsuppgifter för programmeringsstudenter, för att på så sätt få en kontrollgrupp att jämföra med. Dakhel m.fl. (2022) kommer fram till att Copilot löser fundamentala algoritm-problem på ett tillfredsställande sätt, men att mer komplicerade problem är mer benägna att innehålla buggar eller kod som inte löser uppgiften. Författarna menar att den genererade koden också till stor del beror på hur prompten till Copilot är utformad. Däremot kan koden, även om den inte är komplett, användas av systemutvecklaren med viss korrigerings för att ändå spara in värdefull tid. Finnie-Ansley m.fl. (2022) kommer även de fram till att Copilot behärskar problem av lägre komplexitet, och uttrycker att resultaten av genererad kod är bättre än första-års-studenter av programmering. Finnie-Ansley m.fl. (2023) utforskar i en nyare studie hur Copilot klarar av

mer komplicerade problem gällande datastrukturer och algoritmer, och finner att det då genereras en större vidd av lösningar. Det visar sig också vara tydligare att Copilot kan ge mer felaktiga förslag när problemet är mer komplicerat, dock är det fortfarande en stor del av förslagen som är rätt eller bara har små felaktigheter. Problemen motsvarade uppgifter som avancerade studenter får, och Copilot presterade bland de bästa 25% av studenterna som agerade kontrollgrupp. I spekulationer kring vilken typ av uppgifter som passar Copilot, så säger Finnie-Ansley m.fl. (2023) att komplexa problem med svår- eller odefinierade begränsningar medförde sämre kvalitet på kodförslag.

2.3.2 Genererad kod och läsbarhet

Al Madi (2022) har undersökt hur läsbar kod genererad av Copilot är genom att mäta hur lång tid som datavetenskaps-studenter spenderar på att läsa mänskligt skrivna respektive Copilots genererade kod. Författaren kommer i studien fram till att Copilots kod tar kortare tid att granska, vilket spekuleras skulle kunna tyda på hög läsbarhet. Nguyen & Nadi (2022) styrker detta genom en kvantitativ metod, och bekräftar att Copilot oftast genererar läsbar kod, men att det kan innebära att koden är mindre kondenserad. En annan aspekt kring AI tas upp av Al Madi (2022), *automations-bias*, vilket förklaras som att människor har en tendens att lita på automatiska processer. Detta då några deltagare observerades acceptera kodförslag från Copilot utan att granska dem över huvud taget. Al Madi (2022) fortsätter att diskutera att blint acceptera kod kan leda till problem längre fram i utvecklingsstadiet, men också kring fördelen att systemutvecklare får mer tid att granska kod vilket benämns som en av de viktigaste uppgifterna de har. Chen m.fl. (2021) hyser samma oro kring *automations-bias*, speciellt för oerfarna systemutvecklare. Författarna spekulerar även om att ju bättre Copilot blir på att generera fungerande kod, desto svårare blir problemet kring granskning.

2.4 Parprogrammering

Parprogrammering innebär att två systemutvecklare arbetar tillsammans där den ena tar rollen som förare och skriver koden och den andra tar rollen som navigerare och styr riktningen på arbetet samt granskar koden i realtid (Bird m.fl., 2022). *Mob-programming* fyller en liknande funktion men här arbetar ett helt team (fler än två personer) på samma kod och mer specifikt skall detta arbete också utföras samtidigt på samma plats och vid samma dator (Shiraishi m.fl., 2019). Övergripande för både par- och *mob-programmering* är att rollerna skiftas under arbetets gång för att låta samtliga deltagare ta rollen som navigerare och förare (Shiraishi m.fl., 2019; Williams m.fl., 2000).

Parprogrammering har fått allt större uppmärksamhet genom nya agila arbetsmetoder som *eXtreme Programming* där parprogrammering fyller en stor roll (Beck,

2000; Hannay m.fl., 2009). Tidiga studier visar på att systemutvecklare som parprogrammerar producerar mer läsbar och funktionell kod samt att personerna upplever högre glädje och självsäkerhet gentemot systemutvecklare som jobbar ensamma (Nosek, 1998; Williams m.fl., 2000).

Fördelarna med parprogrammering har dock varit omdiskuterade och det finns många studier som både visar på för- och nackdelar med parprogrammering (Hannay m.fl., 2009). Hannay m.fl. (2009) undersöker tidigare forskning gällande parprogrammering och presenterar fördelar samt nackdelar ur ett mer nyanserat perspektiv. De tar hänsyn till fler variabler, som bland annat tidigare erfarenhet hos utvecklarna, komplexitet av uppgiften samt ansträngning. Studien visar på att parprogrammering leder till kod med högre kvalitet om komplexiteten av uppgiften är hög. Även om det tar längre tid samt skapar en högre ansträngning hos systemutvecklare att jobba på detta sätt gentemot att programmera ensam. Uppgifter med lägre komplexitet går dock snabbare men med lägre kodkvalitet (Hannay m.fl., 2009). Den största nyttan av parprogrammering verkar till stor del komma från verbaliseringen av tankar och idéer, vilket flera av de undersökta studierna kommer fram till (Hannay m.fl., 2009).

I en senare studie har Imai (2022) jämfört parprogrammering och programmering med Copilot där hen finner att arbetssättet med Copilot har en högre produktivitet men sämre kvalitet på koden gentemot om två systemutvecklare sitter tillsammans. Författarens hypotes till detta skulle vara ett överförtroende till AI-verktyg vilket medför att koden inte granskas i samma utsträckning som vid vanlig parprogrammering. Därför menar Imai (2022) att Copilot inte fyller syftet och rollen för mänsklig parprogrammering. Det finns följaktligen ytterligare faktorer att utvärdera när det gäller jämförandet av Copilot och mänsklig parprogrammering. Eftersom Copilot är den första AI-parprogrammeraren som har lanserats på marknaden finns det begränsad mängd forskning tillgänglig för tillfället inom detta område (Bird m.fl., 2022).

2.5 ChatGPT

ChatGPT är likt Copilot en mjukvara som bygger på GPT-3. Den skiljer sig från Copilot genom att den är vidareutvecklad med InstructGPT vilket är en modell för att följa instruktioner och ge svar på ett mänskligt sätt (*Introducing ChatGPT*, u.å.).


Instruktionerna ger man genom ett chatliknande interface där man kan ställa frågor eller beskriva ett problem som man vill ha hjälp med att lösa (*Introducing ChatGPT*, u.å.) (se figur 3). Bull & Kharrufa (2023) finner i sin studie att konversationsförmågan med ChatGPT är en styrka vid programmeringsproblem. Att kontexten av tidigare frågor och svar bibehålls i chatten är en stor del av detta. Detta innebär att man kan be ChatGPT lösa ett programmeringsproblem, och blir svaret inte helt rätt kan man be programmet att justera svaret med en ny prompt. Författarna menar även

att ChatGPT har möjlighet att förklara och rätta kod. Då skickar användaren in kod som meddelande och ber programmet ge en förklaring (Bull & Kharrufa, 2023).

Figur 3

ChatGPT ger ett förslag på kod, och förklarar förslaget, enligt en given prompt.

 Hur skriver jag en funktion i Python som översätter text till rövarspråket?

 För att skapa en funktion i Python som översätter text till rövarspråket kan du använda följande kod:

```
python Copy code
def oversatt_till_rovarspraket(text):
    vowels = "AEIOUYÅÄÖaeiouyåäö"
    translated_text = ""

    for char in text:
        if char.isalpha() and char not in vowels:
            translated_text += char + "o" + char.lower()
        else:
            translated_text += char

    return translated_text
```

I den här funktionen används en loop för att iterera över varje tecken i texten. Om tecknet är en bokstav och inte en vokal, lägger funktionen till bokstaven "o" och sedan samma bokstav i små bokstäver i den översatta texten. Annars lägger den bara till tecknet som det är.

Exempel på hur du använder funktionen:

```
python Copy code
text = input("Ange texten som du vill översätta: ")
oversatt_text = oversatt_till_rovarspraket(text)
print("Översatt text:", oversatt_text)
```

I det här exemplet tar funktionen in en text som användaren anger och skriver ut den översatta texten i rövarspråket.

Vänligen observera att denna implementation inte hanterar specialtecken eller särskilda fall som förekommer i vissa versioner av rövarspråket. Detta är en grundläggande implementation för att översätta vanlig text till rövarspråket.

Regenerate response

Send a message. 

3 Metod

Detta avsnitt kommer att redovisa vald metod, urval och datainsamlingsmetod för studien. Vidare kommer dess genomförande att presenteras samt hur etiska aspekter har tagits i beaktning och hur analysen av datainsamlingen genomförts.

3.1 Litteratursökning

För att skapa en grundläggande förståelse för studiens fält genomfördes en litteratursökning efter relevant information. Litteratursökningen ligger således inte till grund för studiens empiriska resultat. Sökningen har genomförts på sökplatser och databaser, genom Scopus, Digital Library ACM och Arxiv. Då Copilot är ett relativt nytt verktyg så finns det därav begränsad tidigare forskning inom området.

Sökord: "Github Copilot", "AI Code Generation", "Pair programming", "Extreme programming", "Agile methods", "Mob programming", "Codex".

Majoriteten av materialet som använts i studien är konferens- och tidsskriftartiklar. Några av artiklarna är dock så pass nya att de är *pre-print*, alltså ännu inte kritiskt granskade. Artiklarna valdes ändå att tas med i studien, just på grund av det begränsade utbudet av tidigare forskning. Totalt hittades 24 artiklar som ansågs relevanta för att underbygga studien. Dessa artiklar delades in i kategorierna användande, kvalitet samt effektivitet och sammanställningen av dessa beskrivs under avsnittet för bakgrund.

3.2 Angreppssätt

Denna studie är strukturerad som en kvalitativ, explorativ studie. Studien undersöker ett antal seniora systemutvecklare professionella användande av Copilot och analyserar deras upplevelser och erfarenheter. Patel & Davidsson (2019) menar att ett explorativt angreppssätt passar bra när det saknas tidigare forskning inom området, vilket det gör för just studiens inriktning på professionellt användande. Författarna menar att syftet med en explorativ studie därför blir att skaffa sig så bred kunskap som möjligt inom det bestämda området för att kunna beskriva detta grundligt. För att kunna fånga in denna breda bild har semistrukturerade intervjuer valts ut som datakälla. Detta kommer att bidra med en fördjupad bild för att kunna fånga in så många olika faktorer som möjligt (Patel & Davidsson, 2019).

Patel & Davidsson (2019) menar dock att studier där urvalet inte är slumpmässigt utvalt utifrån den population som undersöks, inte kan anses som generaliserbart.

Författarna menar därför att man snarare bör se resultatet som relaterbart och menar på att detaljer kring studien ska vara så pass specificerade och relevanta att det för en person motsvarande ett av intervjuobjekten ska vara lätt att relatera till resultatet. Därför syftar studien till att hitta och diskutera nyanser samt åsikter från den valda målgruppen, och att resultatet av detta sedan kan ses som något att relatera till för läsaren.

3.3 Empiriskt urval

Studiens karaktär ställer kravet att respondenterna idag ska dels vara aktiva professionella systemutvecklare, alltså programmera som yrke, dels ha erfarenhet av att använda Copilot (*professionell / SAOB*, u.å.). Att respondenterna skulle ha erfarenhet av att använda Copilot ansågs viktigt för att kunna få en så nyanserad bild som möjligt och undvika spekulationer. Av samma anledning valdes att avgränsa studien till att intervjua seniora systemutvecklare. Senior är ett svårdefinierat begrepp (*Systemutvecklare*, u.å.). För studien valdes därför att se till arbetserfarenheten, minst fyra år som systemutvecklare, då det kan ses som en indikation för senioritet (*Systemutvecklare*, u.å.).

Genom att välja professionella som målgrupp går det undersöka hur användandet av Copilot är i en verklig produktionsmiljö. Flertalet studier har tidigare undersökt Copilot genom att observera och undersöka testpersoner som fått i uppgift att lösa enskilda, väldefinierade koduppgifter (Barke m.fl., 2022; Dakhel m.fl., 2022; Denny m.fl., 2022; Finnie-Ansley m.fl., 2022, 2023; Vaithilingam m.fl., 2022). Det kan antas att det är skillnad att lösa enstaka programmeringsuppgifter jämfört med att skriva kod som skall vara produktionssäker och i en större professionell kontext. Flertalet av testpersonerna i dessa studier har dessutom varit kopplade till universitetsmiljön och kan därför ifrågasättas i sin relevans gentemot professionellt användande.

För att nå ut till relevanta respondenter har olika metoder för urval använts med hänsyn till att Copilot är en relativt ny teknik samt att studiens inriktning skapar en snäv målgrupp. Initialt kontaktades respondenter inom vår närhet som kunde tänkas uppfylla studiens krav. Dessa personer blev i sin tur ombudade att kontakta personer i deras närhet som eventuellt också uppfyller kraven. Vidare kontaktades också respondenter via grupper som kan tänkas innehålla rätt målgrupp på Facebook och Discord. Angreppssättet medförde till slut att personer med olika typer av roller och bakgrunder medverkade i studien, vilket enligt Myers & Newman (2007) är fördelaktigt för att undvika bias. Personer med geografisk närhet primerades då fysiska intervjuer kan anses fördelaktiga gentemot digitala, eftersom att en geografisk spridning inte hade stärkt studien då en kvalitativ studie ändå inte blir generaliserbar (Patel & Davidsson, 2019).

Vid den initiala kontakten via mail (se bilaga 3) informerades samtliga kontaktade om studiens syfte samt deras roll vid eventuellt deltagande samt att de inte har några förpliktelser och har möjlighet att dra sig ur när de vill (*Ethical Guidelines for Online Interviews / Virtual Ethnographic Methods / Class Research Portfolio*, u.å.). Nedan presenteras Tabell 1 som är en sammanställning över de tolv respondenter som deltagit i studien. Respondenterna har valts utifrån deras nuvarande arbetsroll, professionell erfarenhet, hur lång tid de använt Copilot och vilket primära språk de kodar i.

	Arbetsroll	Erfarenhet	Användning Copilot	Primärt språk
Respondent 1	Konsult. Fokus på backend-utveckling.	15 år	10 månader	Typescript
Respondent 2	Jobbar på ett produktbolag. Fokus på infrastruktur	12 år	18 månader	C-Sharp
Respondent 3	Konsult. Fokus på infrastruktur	4 år	2 månader	Java
Respondent 4	Jobbar på ett produktbolag. Fokus på infrastruktur och frontend	6 år	8 månader	Typescript, C-Sharp
Respondent 5	Jobbar på ett produktbolag. Fokus på både frontend och backend	16 år	18 månader	Typescript
Respondent 6	Jobbar på ett produktbolag. Fokus på infrastruktur	16 år	3 månader	Typescript, C-Sharp
Respondent 7	Jobbar på ett produktbolag. Fokus på integration	13 år	2 månader	C-Sharp
Respondent 8	Jobbar med refaktorering av legacykod	4 år	2 månader	C-Sharp
Respondent 9	Fokus på att ta fram prototyper och verktyg	10 år	18 månader	Javascript, Python, Dart
Respondent 10	Konsult. Fokus på backend	5 år	12 månader	Java, PHP
Respondent 11	Jobbar på ett produktbolag. Fokus på infrastruktur	8 år	1 månad	Typescript, C-Sharp

Tabell 1. Sammanställning av respondenter

3.4 Datainsamling

Studiens empiri ligger i den kvalitativa data som samlats in genom semistrukturerade intervjuer. Patel & Davidsson (2019) menar att öppna frågor ger respondenter möjlighet att själva värdera vad som är problematiskt och med egna ord beskriva för- och nackdelar. Kvalitativa intervjuer lämpar sig bra för att fånga personens egna upplevelser (Walsham, 1995). För att i ett tidigt skede kunna utvärdera och utveckla intervjufrågorna testades intervjumallen genom en pilotintervju och gav på så sätt utrymme att tidigt redigera, ta bort eller lägga till frågor (Bell, 2016; Patel & Davidsson, 2019). Denna intervju valdes senare att ingå i studien då inga ändringar på intervjuguiden gjordes samt att svaren ansågs berikande för studien.

Intervjuerna har i de fall det varit möjligt genomförts fysiskt vilket innebär att de som intervjuar lättare kan observera mimik och kroppsspråk och på så vis fånga upp nyanser som annars är svårt. Observation kan komplettera den informationen som samlats in genom enbart samtal (Patel & Davidsson, 2019). I annat fall har de genomförts på distans via Microsoft Teams. Sammantaget genomfördes tolv intervjuer varav sju var på plats. Intervjuerna pågick i cirka 45–60 minuter. Samtliga intervjuer som skedde på plats utgick från respondentens arbetsplats, något som kan medföra att respondenterna känner sig mer bekväma och därav ger rikare svar (Patel & Davidsson, 2019). Att intervjuerna hölls på respondenternas arbetsplatser var också med hänseende att underlätta och ta deras tid i beaktning (Bell, 2016). Intervjuerna genomfördes av två eller tre av uppsatsskrivarna med olika roller, en med syftet att interagera och ställa frågor och en eller två med syftet att observera, anteckna och eventuellt ställa följdfrågor (Bell, 2016; Patel & Davidsson, 2019). Patel & Davidsson (2019) belyser också vikten av att förhålla sig till forskningsetiska riktlinjer och respondenterna informerades därför inledningsvis i intervjuerna om konfidentialitet i studien samt tillfrågades om tillåtelse att spela in intervjun med avseende att användas endast för transkribering. Respondenterna fick också information gällande att lagringen av denna data kommer att ske säkert via universitets molntjänst och kommer att destrueras efter att uppsatsen är examinerad och godkänd.

Intervjuguiden är indelad i fyra huvuddelar; syfte med studien, bakgrund, användande av Copilot och åsikter om Copilot (se bilaga 1). Bakgrundsfrågorna ger respondenterna möjlighet att beskriva sin erfarenhet som systemutvecklare, arbetsroll samt relation till programmering i syfte att eventuellt senare kunna knyta an i analysen till användandet och åsikterna. Vidare ställs öppna frågor gällande användande, upplevelser och åsikter kring Copilot med stort utrymme för respondenterna att tolka

och svara utifrån egna erfarenheter. Avslutningsvis har olika ämnen som ansetts extra intressanta utifrån tidigare forskning punktats ner som stöd under intervjun. I takt med att vi genomförde intervjuerna tillkom även punkter som ansågs berikande.

Exempel på teman från tidigare forskning kan vara *accelererande, undersökande* samt kvalitet (Barke m.fl., 2022; Jaworski & Piotrkowski, 2023). Dessa punkter har således fyllts på under datainsamlingens gång när fler intressanta och relevanta ämnen har framkommit under intervjuerna. I de fall de inte berörts under intervjun har frågorna tagits upp i slutet av intervjun för att låta respondenten i största mån själv komma in på dessa punkter utan att bli färgad av intervjuaren. En punkt som togs upp i samtliga intervjuer var ChatGPT. Därför har vi valt att behandla det i vår studie. Att börja med öppna frågor för att sedan övergå till specifika frågor menar Patel & Davidsson (2019) kan verka uppmuntrande och hjälpa respondenter att formulera sina tankar och ge dem mer frihet att formulera sig i ett tidigt stadiet. Att bygga upp intervjuerna semistrukturerat utifrån tidigare teman skapar också en struktur att utgå från vilket kan göra analysprocessen enklare (Patel & Davidsson, 2019).

3.5 Analys av data

Det insamlade materialet har analyserats med hjälp av en tematisk analys. Braun & Clarke (2006) menar att tematisk analys är en flexibel metod för att identifiera, analysera och gruppera teman inom kvalitativt insamlade data. Författarna beskriver att tematisk analys går att utföras både deduktivt och induktivt. Studiens ansats har utförts induktiv då forskningen är mycket begränsad gällande studiens syfte och därav ansågs det viktigt att kunna fånga upp en så bred och nyanserad bild som möjligt utifrån intervjupersonernas syn och erfarenhet (Patel & Davidsson, 2019). Intervjuerna transkriberades löpande med hjälp av Whisper, som är en AI-modell för att översätta tal till text (*Introducing Whisper*, u.å.). För att säkerställa kvalitén lyssnades samtliga inspelningar igenom parallellt med att transkriberingen lästes för att kontrollera att dessa var korrekta, och de meningar och ord som var felaktiga rättades manuellt. Transkriberingen resulterade i cirka 120 sidor material. För att analysera och strukturera upp detta material användes Atlas.ti, ett mjukvaruprogram för textanalys (*ATLAS.Ti*, u.å.).

Den inledande kodningen utfördes med ett semantiskt tillvägagångssätt i syfte att organisera det insamlade materialet utan att tolka och analysera eventuella underliggande meningar. Dessa övergripande koder skapades till stor del utifrån intervjuguiden men också från andra intressanta teman som framkommit. Vidare delades dessa koder in i underkategorier för att enklare kunna jämföra liknande förekomster över flera intervjuer (se bilaga 2). Utifrån dessa koder skapades övergripande teman med ett latent tillvägagångssätt. Dessa teman granskades sedan och överlappande teman

slogs ihop samt teman som inte ansågs direkt kopplade till studiens syfte sorterades bort. De slutgiltiga temana presenteras i resultatavsnittet.

4 Resultat

Resultatet av analysen presenteras utifrån sex övergripande teman med underkategorisering. Det första avsnittet handlar om respondenternas bakgrund och deras relation till programmering. Detta avsnitt ämnar ge en grund till att förstå respondenternas svar i vidare delar i resultatet. Fortsättningsvis presenteras avsnitt gällande hur och varför respondenterna använder Copilot och följer med vilka nyttoeffekter som kommer med användandet. Vidare presenteras ett avsnitt gällande ChatGPT då detta var ett ämne som framkom frekvent i samtliga intervjuer och ansågs viktigt att lyfta fram i studien eftersom tjänsten starkt relaterar till och kompletterar delar av Copilot. Avslutningsvis presenteras ett avsnitt gällande respondenternas syn på likheter mellan Copilot och parprogrammering.

4.1 Respondenternas relation till programmering

4.1.1 Programmering en del i verktygslådan

Det finns olika nyanser bland respondenterna gällande huruvida programmering är en passion och hur deras relation ser ut till programmering. Flertalet respondenter uttrycker att relationen till att programmera har förändrats under deras professionella resa och att de betraktar programmering som ett verktyg som används för att utföra sitt professionella arbete, snarare än en huvuduppgift i sig själv. De ser programmering som en del av sin arbetsprocess, men inte nödvändigtvis det primära fokusområdet. Programmering handlar också om problemlösning för många och vägen för att komma fram till en produkt eller lösning på ett övergripande problem.

”Det har väl förändrats också lite med tiden. Men som jag sa det har blivit mer och mer av ett verktyg för att åstadkomma en värdeförbättring för våra kunder. Mer än att det bara är kul tekniklek som man kanske mer ser på det när man har det som hobby. Men här i det professionella så ser jag mer på det som ett verktyg som vi använder för att utföra vårt jobb. Så kanske inte det som är vårt huvudsakliga jobb egentligen att vara programmerare. Utan det är ett av våra verktyg. Vår verktygslåda.” – Respondent 6

4.1.2 Läsbar kod framför optimering

Då flertalet av respondenterna har en lång professionell bakgrund så följer också ett tydligt mönster över att ta ansvar för att skriva bra kod med hög kvalitet men också med eftertanke för att andra lätt ska kunna förstå och kunna läsa koden i framtiden.

”När jag skriver kod tänker jag väldigt mycket på den stackare som ska läsa och ändra koden i framtiden. Så jag försöker skriva så snäll kod som möjligt. För jag har själv under mina 15 år hamnat i läget i projekt där jag tvingas ta över annan kod. Och jag vet hur hemskt det är att ta över kod som är jobbig att läsa.” – Respondent 1

Det finns endast en respondent som i sitt nuvarande arbete skriver kommentarer då det bland övriga respondenter anses överflödigt till följd av att koden bör vara så pass välskriven att inte kommentarer behövs.

”Om du skriver bra och kod så behöver du inga kommentarer. Koden är en kommentar.” – Respondent 11

Även om många respondenter betonar vikten av att skriva kod med hög kvalitet finns det däremot ingen som framhåller vikten av att skriva kod med hög prestanda. Några av de tillfrågade respondenterna förklarar att detta beror på deras specifika arbetsroll och att det kan finnas andra arbetsroller där höga prestandakrav är mer betydande.

Jag jobbar ändå i en värld där saker inte alltid måste optimeras. Hade jag jobbat som en spelprogrammerare (...) så vet jag inte hur det skulle bli.
– Respondent 9

4.1.3 Tråkigt med repetitiva uppgifter

En tydlig uppfattning gällande vad som är tråkigt med programmering är att skriva boilerplate-kod, d.v.s. kod som upprepas ofta och som har en låg variationsgrad. Denna typ av kod anses endast vara tidsödande.

”Det är väl boilerplate-kod. Om ni förstår vad jag menar med det. Rörmokeri. Det är saker som man måste göra fast det är inte där affärsvärdet ligger inkodat.” – Respondent 6

Det hänger starkt ihop med respondenternas erfarenhet och att de redan sedan innan vet exakt hur de ska skriva koden och gjort detta flertal gånger innan och därför inte blir ett givande arbetsmoment utan ett nödvändigt ont.

”Eller när jag måste göra någonting som jag gjort för många gånger så att det börjar bli tråkigt och trist” – Respondent 1

4.1.4 Flow i arbete

Något som uppfattas som roligt och givande beskrivs ofta som när respondenterna kan arbeta utan avbrott. Många benämner det som ett slags *flow* eller *flyt* i arbetet som upplevs som både roligt och tillfredsställande.

”Det roligaste är när man kommer in i flow. Man vet exakt var man ska. Det går sjukt snabbt att komma dit.” – Respondent 5

På samma sätt upplevs också saker som stör detta *flow* som tråkigt och frustrerande. Det kan röra sig om att behöva lämna IDE:n eller bara stöta på förhinder som man inte hade förutspått.

”Men när det kommer till själva skrivandet är det kanske mer när man hamnar i ett flow. Att man bara inte har förhindren. Och bara kan köra på och se vad man gör och fortsätter och modifierar. När man hamnar där så är det riktigt skoj. När man inte är där och hakar upp sig.” – Respondent 9

4.2 Varför började de använda Copilot?

Det finns flera anledningar till att respondenterna började använda Copilot. Nyfikenhet på ny teknik är en drivkraft för många. Nyfikenheten verkar vara grundad i två delar, dels är det flera som uttrycker en stark nyfikenhet på ny teknik, dels vill man se vad de nya språkmodellerna kan åstadkomma. En del av nyfikenheten kommer från den rådande hajpen för generativa AI modeller. Både specifikt för Copilot men nyfikenheten har även ökat med anledning av ChatGPT som fått stor uppmärksamhet den senaste tiden.

”Och där blev det så himla hajpat att det var bra på alla sätt och vis. Och övriga världen började prata om det också. Och då tänkte jag att det måste ju vara någonting man måste prova för att se vad det kan göra.”

– Respondent 7

Flera uttrycker att AI-verktyg är framtiden. I takt med att de generativa modellerna blir effektivare ser fler nyttan i att använda dem. De uttrycker också en viss rädsla i att hamna efter. Respondent 1 uttryckte rädslan för att AI kommer att ta ens jobb *”Jag testade litegrann och upptäckte att shit, den här kommer att göra mig arbetslös.”*. Men de flesta ser snarare att de som vänjer sig vid att använda verktygen kommer att ha en fördel.

”Du ska inte vara rädd för att AI tar ditt jobb utan du ska vara rädd för att någon som kan använda AI tar ditt jobb” – Respondent 4

Enkelheten att komma i gång är också en anledning för många att börja använda Copilot. Det är ett *plugin* som skall installeras och sen är det bara på i bakgrunden. Den generösa gratis provperioden (2 månader) bidrar till lättheten i att testa och komma i gång.

”Det var ju gratis att börja använda det i början. Det var ju låg tröskel till att prova.” – Respondent 12

4.3 Copilots användande

I detta stycke presenteras hur respondenterna använder sig av Copilot. Här presenteras även vad de upplever som begränsningar i verktyget. Avslutningsvis lyfts några fall där organisationer valt att begränsa användandet.

4.3.1 Hur används Copilot?

Det sätt alla respondenter har beskrivit att de använder Copilot på är att det är en avancerad *autocomplete* när de programmerar. De börjar skriva ett stycke så som de tänkt att koden ska se ut och fungera, och Copilot kommer snabbt med ett förslag på fortsatt kod.

”Vanligtvis är det den här autocomplete-funktionen helt enkelt. Den gissar ganska bra vad man vill åt. Så man börjar knacka och så tabbar för att autocompletea och så fixa till det som den, den gör ju inte alltid 100%. Men det är mycket där det är bara autocomplete. Det tycker jag att den gör galant.” – Respondent 7

Många respondenter trycker på att även om förslagen man får från Copilot inte alltid är helt korrekta, så är det oftast bara att manuellt justera felet man har fångat upp.

”Oftast låter jag det bara föreslå vad som ska göras när jag börjar skriva något. Jag kan börja namnge en funktion så fattar den vad jag tänker göra utifrån vad som redan finns i projektet. Ibland gör den fel på en rad. Då får man rätta till den.” – Respondent 12

Ibland så kommer däremot Copilot med förslag som inte går att använda eller inte är värda tiden det tar att justera. När det kommer till hur eventuella oönskade förslag hanteras så har respondenterna två olika sätt att arbeta på. Det första sättet är att ignorera förslaget och fortsätta att skriva som vanligt.

”Jag kollar bara på det första förslaget. (...) Men om jag inte gillar det första så ignorerar jag det förslaget och skriver vidare själv. Om den ändrar sig och förslår något som jag tycker ser rätt ut, då accepterar jag det förslaget.” – Respondent 3

Det andra sättet är att man ändrar något i koden man redan har skrivit, för att få Copilot att ”tänka om”. Ett exempel är att skriva tydliga och beskrivande namn på funktioner eller variabler.

”Då ska jag börja med translationfiler. Då har vi nycklar, och sen så under där så har vi typ sv och så en för engelska varianten. Då har vi ibland bara skrivit nyckelnamnet. Sen har jag gjort förslag både på det svenska och engelska. Där har vi ibland bara formulerat om nyckeln lite grann. Då har den gjort små förändringar i texten och bara varit totalt klockrent... Jag hade

aldrig kunnat skriva den här texten bättre. Man gör lite sådana grejer ibland. Man kan lite tweeka på namnen ibland.” – Respondent 2

En person sticker ut i sitt användande, på det sätt att hen främst använder promptar i naturligt språk för att få förslag på kod. Beroende på komplexitet och storlek på koden så bryter hen ner logiken i mindre steg för att ge mer och tydligare kontext till Copilot. Utgångsläget är fortfarande att kunskapen om hur koden borde se ut och vad den ska göra redan finns hos personen.

”Så länge man bryter ner det i tillräckligt små beståndsdelar så skulle jag alltid säga att det fungerar. (...) Men jag har en idé innan vad det är jag vill göra. Och jag tror också jag har en idé om hur jag kan bryta ner. Om det är en jättekort funktion så kan jag få hela funktionen i en prompt. Men är det en större grej, då skriver jag först funktionen och sen i funktionen lägger jag kommentarer för varje del av det jag vill ha genererat.” – Respondent 9

Flera andra respondenter beskriver att de har testat att använda Copilot på samma sätt, men kommit fram till att det inte har levt upp till deras förväntningar och att *autocomplete* fyller deras behov bättre.

”Det är väl just när den ska spotta ut sig mer komplex kod. Då känner jag att det är ingen anledning att försöka egentligen. Den är inte lika duktig på att ta in kontext i klarspråk. Kontext i kod, det klarar den jättebra. (...) Men försöker du förklara det i text så funkar det inte lika bra.” – Respondent 5

Att skriva promptar i naturligt språk för små funktioner har ett par respondenter nämnt fungerar men så fort problemen blir för stora menar de att Copilot inte löser det så bra. Där tycker de att ChatGPT gör ett bättre jobb.

”För enklare grejer är den jätteduktig. Sortera listan efter nummer. Då kan den spotta ut sig bra. När det börjar bli mer komplexa grejer då är den inte lika duktig som chatGPT.” – Respondent 5

Flera respondenter väljer att inte acceptera för många rader kod åt gången för att på så sätt bibehålla kontroll över koden som genereras.

”Jag låter den generera en till tre rader. Och sen fyller jag på resten själv. Så det blir mer en... Autocomplete. Fast den förstår kontexten. (...) Jag litar inte på den till 100%. Så då vill jag ha kontrollen själv.” – Respondent 5

Den känslan av att bibehålla kontroll är en viktig del i ansvaret för koden. Alla respondenterna uppger att de tar ansvar och kvalitetssäkrar den kod de producerar. Oavsett om den är genererad med Copilot eller skriven för hand.

”Innan jag liksom committar kod så ser jag det som mitt. Det är ju mitt jobb att se till att det inte, att någonting blir fel. Och om Copilot genererar någonting som är fel, då hoppas jag att jag märker det lika mycket som jag hade märkt det om jag själv skrev det.” – Respondent 9

En respondent tar upp att en risk med AI-modeller är att de kan framstå som väldigt självsäkra i sina svar vilket lätt kan misstolkas för att den har rätt, medan de i själva verket endast genererar text och de har inget sätt att validera om outputen är korrekt.

”Man måste ju ha tester för den koden som den spottar ur sig. (...) Just i och med att många av de här modellerna som ChatGPT kan ju ibland ha fel och den vet inte om att den har fel. Den är liksom som man säger confidently wrong.” - Respondent 10

En annan respondent beskriver hur hen skulle kunna använda långa stycken kod genererade av Copilot i specifika fall såsom utveckling av *proof of concept*, där syftet med utvecklingsmetodiken är att det ska gå fort snarare än att hålla samma höga kvalitet.

Men om jag skulle använda Copilot till att skriva större sjok med kod. Så dels är det svårare för mig att med egna ögon kvalitetssäkra den koden som Copilot ger mig. (...) Får jag plötsligt den här bunten med 100 rader eller 50 rader produktionskod. Så behöver jag kanske sätta mig och fundera igenom hur ska jag testa den. (...) Men det är som sagt, det gäller ju den här typen av produktutveckling. Vi utvecklar kod som ska leva i 20 år. Hade jag slutat och gjort kanske en proof of concept som jag sen ska slänga, då hade jag gladeligen kanske tagit den här koden som jag får gratis utan att tänka på det så mycket. – Respondent 6

En vanligt förekommande åsikt är därav att det är nödvändigt att ha en grundläggande förståelse och erfarenhet för korrekt kunna utvärdera och använda det som genereras av Copilot, samt att effektivt redigera och anpassa koden.

”Så den är som en bästa vän som är lite på fyllan. Du kan inte riktigt lita på det den skriver ut. Man får ändå kunna lite kring det.” – Respondent 8

Flera respondenter nämner också att Copilot producerar läsbar kod vilket både kan underlätta granskandet av koden samt att det stämmer överens med hur de själva vill skriva kod.

Vi tittar inte så mycket på, dels på effektivitet (...). Det handlar mer om läsbarhet som är vårt största fokus egentligen. Att det ska vara läsbart efteråt. Och det tycker jag att Copilot gör väldigt bra, att det är läsbar kod. – Respondent 4

4.3.2 Copilots begränsningar

Copilot verkar bland respondenterna enhälligt bra på att generera korta stycken kod. Däremot finns intresset av att kunna använda längre stycken. Här finns dock en enhällighet i att Copilot inte riktigt håller måttet, och det resoneras kring hur man kan nå och använda de längre styckena. En respondent funderar på att hellre använda sig av ChatGPT för detta ändamål.

”Nu sitter jag lite sugen på att jag kanske ska betala de här pengarna för att köpa ChatGPT+. För att i mina ögon, Copilot genererar små bitar kod, men ibland känner jag att jag skulle vilja ha mer kontext och ännu mer genererat.” – Respondent 9

Flera respondenter har också belyst en annan begränsning med Copilot som berör att skriva promptar med naturligt språk. De beskriver det som att den ofta ger helt felaktiga förslag och att det kan vara svårt att veta hur man ska formulera sig för att få rätt förslag.

Den är inte lika duktig på att ta in kontext i klarspråk. Kontext i kod, det klarar den jättebra. Ger du den en typ med olika värden så förstår den vad den kan göra med den typen. Men försöker du förklara det i text så funkar det inte lika bra. – Respondent 5

Många av dessa respondenter lyfter dock upp att de utforskat detta område väldigt lite och att de i ett tidigt skede dömde ut denna funktionalitet utan att utforska den vidare.

Jag misstänker att jag testade det lite i början och tänkte att det här blir ändå inget vettigt. Så har jag liksom glömt av det där lite. – Respondent 1

4.3.3 Reglering av Copilot

Två av respondenterna tar upp reglering i användandet av Copilot i deras organisationer. En av dessa får endast använda Copilot vid programmering av interna verktyg, och det verkar bero på en rädsla för eventuella ägande-tvister som skulle kunna komma på sikt.

”Man visste inte vem som ägde källkoden som den skrev. (...) De har en policy där jag är idag som säger att vi får använda AI-genererad kod för interna tools och interna verktyg och applikationer med allting som externt utåt. Det är för att de inte vet vem som äger det som produceras. Det är en pågående diskussion och undersökning.” – Respondent 3

Den andra respondenten arbetar på en myndighet och har därmed en hög nivå av säkerhetsklassning. Då är det problematiskt att skicka data till servrar som man inte själv har kontroll över.

”Vi får inte använda den (Copilot) på arbetsplatsen än. (...) I och med att det är en myndighet så måste vi säkerhetsklassas.” – Respondent 8

4.4 Nyttoeffekter av Copilot

4.4.1 Utbildande och utmanande

Det är flera som pratar om hur de kan lära sig nya saker av att arbeta med Copilot, och det verkar finnas olika nyanser av lärandet. En del är att man får förslag som avslutar koden på ett sätt som man inte tänkt på. Då lär man sig nya sätt att lösa en uppgift på och kan skapa nya mönster i hur man arbetar. Det utmanar då ens arbetsmönster.

”För annars ligger man ju väldigt lätt i sin bubbla som programmerare. Så här har jag alltid gjort, och så här fortsätter man. Och så löser man samma sak på samma sätt hela tiden. Och det hjälper ju Copilot en, att tänka lite utanför boxen på så sätt.” – Respondent 4

Copilot kan också bidra till en form av lärande när användaren har en konceptuell förståelse för hur man kan lösa ett problem, men kan inte korrekt syntax, exempelvis när man arbetar med ett nytt ramverk eller programmerar på ett nytt språk. Copilot kan då minska behovet av att söka efter information och resurser på webben. Därmed kan inläringen överföras från att lära genom exempelvis forum på webben till att Copilot fungerar som en sorts lärare. Detta gäller dock främst för rena kodproblem, medan andra typer av problem såsom hårdvaru- eller systemproblem fortfarande kan kräva ytterligare sökning av information och kunskap på webben.

”I och med att man skiftar så mycket mellan språk så kan det vara att man behöver refresha minnet för att skapa en lista. Det är sådana vanliga grejer som man oftast behöver googla och då kommer man oftast in på stackoverflow. Men sen så ifall man har problem med hårdvara eller något med systemet i sig, då googlar jag för uppförande och då kommer jag oftast in på stackoverflow. Men kodmässigt så har de här vardagliga grejerna blivit betydligt enklare för det föreslår Copilot så att man inte behöver tänka på hur man genererar en lista” – Respondent 3

Att skriva tydlig kod är viktigt för flera av respondenterna. Det innefattas även namngivning av funktioner och variabelnamn. Eftersom Copilot kan generera kod utifrån funktionsnamnen kan man där få en hint om namnet man valt är ”bra” beroende av den koden Copilot föreslår. Copilot kan då bidra till att man tänker till en extra gång och utvecklar sitt skrivande till att bli mer tydligt.

”Man får man oftast kodförslag efter man har skrivit namnet på funktionen. Det man kan göra då är att tänka efter om det är fel förslag. Då har jag nog

satt fel namn på funktionen också. Det kan ju vara en sån sak. Så det blir ju lite sån eftertänksamhet där.” – Respondent 4

4.4.2 Minskad kognitiv ansträngning

Uppgifter som är repetitiva är en del där man kan spara mycket tid. Det kan både vara uppgifter som ingår i varje projekt, *boilerplate*-kod, och i sammanhang där man skall utföra repetitiva uppgifter som att skriva tester. Detta är också något som framkommit som tråkiga moment av programmering från respondenterna. En stor del i att använda Copilot är att det går fortare att skriva koden. För vissa är det en vinning i sig att det går fortare. Det gör att arbetet flyter på bättre och känns roligare.

”Jag behöver ägna mindre tid åt den här grejen. Jag sparar tid. (...) Det är att man kan bespara sig själv en viss typ av... Alltså tankekraft och tid.” – Respondent 2

Några beskriver hur kodskrivandet är en förhållandevis liten del av arbetet och för flera är det bara ett verktyg. Då är den ökade hastigheten inte lika viktig men de anser fortfarande att det bidrar till att frigöra utrymme för att tänka på det som är mer väsentligt.

”Eftersom kodskrivandet är en liten del av den totala tiden man lägger ner så är den begränsad. Det är ungefär som någon som är flink vid tangentbordet jämfört med någon som sitter med pekfingerens. Det blir inte så stor skillnad i längden. (...) Det kan vara mer, kanske arbetsmiljömässigt, att det blir mindre frustration i det dagliga arbetet. För att man inte behöver sitta och brottas med den här boilerplate-koden. Man sparar sina kognitiva resurser i hjärnan till andra saker. Det kan ändå ge större effekter än man kan tro.” – Respondent 6

4.4.3 Paradigmskifte inom programmering

Det är också flera som jämför användandet av Copilot med att byta till en högre nivå av språk. Det har hänt flera gånger förut och det följs ofta av diskussioner kring huruvida nya systemutvecklare kommer kunna hantera det.

Det blir bara en högre nivå av kod, men det är ju fortfarande kod. Så vi kommer aldrig att komma bort ifrån kod egentligen. För det har ju alltid varit så att vi har ju höjt nivån hela tiden. Jag menar från början så programmerar man ju bokstavligen ettor och nollor. – Respondent 10

Flera av respondenterna ser det som en fördel att slippa komma ihåg detaljer från ramverk eller bibliotek. Med stor erfarenhet av programmering och tidigare användning av liknande bibliotek är det inte längre nödvändigt att lära sig då Copilot nu kan tillhandahålla detta.

”Då började jag bara veta utifrån erfarenhet vad jag ville göra med den. Då fick jag väldigt mycket hjälp och skjuts av att den föreslog hur jag ska fylla på. (...) Om jag har Copilot så behöver jag inte komma ihåg de här sakerna.” – Respondent 12

4.5 ChatGPT, ett komplement till Copilot

Flera av respondenterna anger att de använder ChatGPT till olika programmeringsrelaterade arbetsuppgifter. ChatGPT kan till exempel användas för att ge en generell idé om hur man skall lägga upp sitt program, där man sen skriver koden med hjälp av Copilot.

”Det blir en liten kombination av ChatGPT och Copilot där de blir en samverkan. ChatGPT ger mig lösningen på problemet medan Copilot blir den som hjälper mig skriva koden sen.” – Respondent 4

Men även att man använder ChatGPT för refaktorering, felsökning samt att få kod förklarad. En person uppger att hen promptar ChatGPT för att få ett lösningsförslag som de sedan refaktorerar med hjälp av Copilot.

”Jag har kanske inte använt den fullt ut Copilot. För jag har inte sett någon möjlighet att skicka in promptar precis som man gör till ChatGPT på samma sätt till Copilot. För att säga att jag vill refaktorer en metod, få en metodförklaring eller felsöka.” – Respondent 3

En annan anledning till att använda sig av ChatGPT framför Copilot berör det mer naturliga sättet att kommunicera med verktyget genom dess chattfunktion. Respondent 4 benämner det som att det känns mer naturligt att kunna resonera och ställa frågor för att komma fram till ett resultat gentemot att bara försöka skriva om prompts tills att resultatet blir rätt. Detta arbetssätt lämpar sig dock endast för när man vill lösa mer avancerade problem eller ha större stycken kod.

Det som är bra med chattande på så sätt är att du kan få, (...) man kan ju mer ”rätta den” och sånt. Så om man har, ska göra något ganska stort avancerat och så kan man ju mer börja med en fråga och så ger den ett svar och så säger det blir mer att kommunicera med den för att få fram ett svar som är passande. Det blir svårt med Copilot att rätta den på så sätt.
– Respondent 4

Respondent 10 benämner det som att man kan se ChatGPT som ett komplement till Copilot. ChatGPT fyller då de luckor som finns hos Copilot gällande diskussion och kan generera stora stycken kod.

Jag tycker nog att de är lite grann av ett komplement till varandra snarare än att de gör samma saker. Jag vill ju inte använda ChatGPT till code completions för det är för långsamt. ChatGPT upplever jag att där kan jag mera ställa frågor på hur jag ska lösa saker när jag inte vet hur jag borde lösa dem. Medan Copilot är mer att fullfölja det som jag redan har påbörjat, som jag redan vet hur jag skulle ha löst. – Respondent 10

4.6 Copilot som parprogrammerare

Flera av respondenterna arbetar med par- eller mob-programmering och beskriver att syftet till den arbetsmetoden är att vara två eller flera hjärnor som kompletterar varandra. Det handlar mycket om att kunna diskutera fram lösningen i ett större perspektiv.

”När man gör parprogrammering så är man ju verkligen två hjärnor som tänker på samma sätt. Som jobbar på ett liknande vis.” – Respondent 12

Syftet med par- och mob-programmering handlar enligt respondenterna inte om att uppnå snabbare resultat, utan snarare att producera högkvalitativ kod och främja en naturlig diskussionsplattform där alla inblandade är fokuserade på samma uppgift. En fördel med att arbeta fler tillsammans på samma uppgift är att det kan vara svårt att själv granska sin egen kod och att flera ögon från olika perspektiv kan hjälpa till att upptäcka eventuella fel och förbättringar. Denna samarbetsmetod bidrar således till högre kvalitet på den slutliga koden men nödvändigtvis inte att det går snabbare.

”För jag ser inte parprogrammering som att man skriver kod snabbare. Dels att du har ett par extra ögon. Det är en person som skriver kod och så har du ett par extra ögon som hjälper dig hitta fel.” – Respondent 9

Respondenterna uppger att Copilot inte riktigt kan fylla den rollen än. De beskriver det som ett mer ojämnt förhållande där de utför sina sysslor och Copilot hjälper till så gott den kan.

”Parprogrammering kanske handlar mer om att ha olika idéer och bolla idéer fram och tillbaka. Men Copilot känns mer som att man ger någon order om att göra någonting. Och så säger man nej, det var fel. Så det är lite mer ett slaviskt förhållande.” – Respondent 7

Flera respondenter lyfter också att de dagligen jobbar med Mob-programmering och att detta arbetssätt innebär en del väntan, framför allt i samband med att boilerplate kod skall skrivas. En respondent menar därför att använda sig av Copilot i samband med mob-programmering då skulle kunna minska en del av väntan och på så vis höja engagemanget.

”Det blir extra påtagligt i mobben när man måste skriva tråk-kod. Då sjunker engagemanget och man måste sitta och vänta. Det är den perfekta kombon egentligen. Sammanlätt så har man snabbare kommit till problemets kärna. Det är då man behöver varandra.” – Respondent 6

En respondent tar upp att hen använder ChatGPT som komplement till Copilot och att de tillsammans mer kan fylla rollen som en parprogrammerare.

”Det som den saknar i parprogrammeringsbiten, det är ju mer diskussionsbitar vilket blir svårt. Det är lite där som chatGPT kan komma in i stället som en sån roll”. – Respondent 4

5 Diskussion

Detta kapitel diskuterar användandet av Copilot hos seniora systemutvecklare gentemot tidigare forskning. Här behandlas hur respondenterna använder sig av Copilot och vilka delar som inte är lika frekvent använda. Vidare diskuteras deras syn på ansvar och hur de ser på AI-genererad kod samt hur Copilot står sig som parprogrammerare. Avslutningsvis diskuteras kvaliteten på koden genererad av Copilot och hur det påverkar användandet.

5.1 Effektivare arbete med Copilot

Barke m.fl. (2022) delar in nyttjandet av Copilot i två delar, *accelererande* och *utforskande*, vilket vi menar stämmer överens med respondenternas beskrivning. Vi finner att alla använder Copilot på ett sätt som stämmer in på *accelererande*. De vet redan innan de börjar vad de skall skriva och använder därför Copilot som en mycket sofistikerad *autocomplete*. Det är för många en stor vinning i sig, framför allt när man skriver så kallad ”boilerplate” kod, vilket är samstämmigt med Barke m.fl. (2022) och Sarkar m.fl. (2022). Respondenterna lyfter framför allt hur det bidrar genom att minska det tråkiga och repetitiva arbetet. Eftersom respondenterna också har en tydlig bild om vad de vill skriva kan de också ofta snabbt se om det förslaget de får från Copilot är vad de tänkt sig, och är det inte det kan de fortsätta skriva tills Copilot fattat vad som skall göras och sen accepterar de förslagen.

Det *utforskande* arbetssättet är av Barke m.fl. (2022) definierat som när man inte vet hur man skall lösa en uppgift. Det är tydligt att våra respondenter får fördelarna av detta arbetssätt. När de tar sig an ett nytt ramverk eller ett nytt språk har de en så pass god grundförståelse att de kan börja skriva och får relevanta förslag genom att göra som man tror att det borde se ut. Det bidrar till ökat *flow* när de arbetar och de slipper lämna IDE:n för att exempelvis söka specifika metoder i ramverk eller läsa på syntax i ett nytt språk.

Flera respondenter tar också upp hur programmering endast är ett verktyg för att lösa ett problem. De ser sig själva mer som produktutvecklare än programmerare. För många är det en förhållandevis liten del av deras arbete som består av programmering men det är också avlastande att effektivisera den delen. Som Barke m.fl. (2022) också tar upp blir det en kognitiv avlastning som frigör tankekraft åt andra arbetsuppgifter.

5.2 Promptar i naturligt språk: det utforskade arbets-sättet

Det är tydligt att skriva promptar i naturligt språk i Copilot är något som respondenterna för studien i huvudsak inte har anammat. Endast en person använder promptar på detta sätt aktivt i sitt arbete, som sitt huvudsakliga tillvägagångssätt. Intressant nog beskriver också personen att hen arbetade på samma sätt innan Copilot började användas, alltså att försöka dela upp problem i mindre delar och skriva kommentarer eller pseudo-kod för att skapa struktur. Det tillvägagångssättet tar flera andra respondenter upp som något de knappt ens testat, eller testat och funnit ”onaturligt” för deras arbetsflöde. När de tillfrågats om de brukar skriva kommentarer i sin kod har svaret i alla fall utom ett varit att kommentarer är onödiga i kod och om man behöver skriva dem så är koden dåligt skriven. Koden ska alltså vara självförklarande och man ska inte behöva lägga tid på att förtydliga med naturligt språk. Detta krockar med hur funktionen att prompta Copilot med naturligt språk är utformad. Extra tydligt blir detta när respondenterna löpande under intervjuerna avslöjar att de gärna tillfrågar ChatGPT i naturligt språk kring mer komplicerade frågor som de säger sig uppleva att Copilot inte klarar av. Detta då utan att ha vidare utforskat promptar i form av kod-kommentarer till Copilot. Man kan därför anta att många av de exempel som lyfts av respondenterna gällande hur de använder Copilot inte nödvändigtvis hänger ihop med det sätt som skulle ge dem mest nytta, utan snarare hänger ihop med det sätt som mest liknar det arbetssätt de är vana vid att jobba med sedan innan. Det kan också antas hänga ihop med att tröskeln för att använda Copilot har ansetts låg av respondenterna och det därför är lätt att endast utforska de arbetssätt som faller sig naturligt. Uppenbarligen finns ju nytta där ändå, då alla respondenter uttrycker att de är väldigt nöjda med Copilot.

En komponent i att framgångsrikt arbeta med promptar i naturligt språk verkar vara att dela upp problemet i små uppgifter vilket stöds av Barke m.fl. (2022). Flera respondenter uppger att de tycker att Copilot har svårt att generera större stycken kod, så det faller sig naturligt att mindre delar fungerar bättre. Längden på promptar tycks också vara en indikation på hur väl förslaget stämmer med förfrågan, vilket är samstämmigt med vad Denny m.fl. (2022) kommer fram till. Kan man inte beskriva uppgiften i en mening så kommer Copilot förmodligen inte lyckas generera ett speciellt bra svar. Det stämmer även in på de förslagen man får efter att bara namngivit en metod. Är namnet bra får man ofta en bra lösning. Det verkar som att man behöver lära sig hur man ska formulera prompts med naturligt språk för att få ut bra resultat, vilket kan vara anledningen till att majoriteten av våra respondenter tidigt avfärdade detta arbetssätt då de inte har haft direkt erfarenhet av detta innan och resultatet antagligen därav blivit sämre.

Barke m.fl. (2022) beskriver att medverkande i deras studie använder promptar i naturligt språk, och att man då antingen får ”gratis” kommentarer som går att använda som dokumentation eller att man helt enkelt tar bort kommentaren efter att koden genererats. Att få ”gratis” kommentarer är inte eftersträvansvärt hos respondenterna. Snarare tar man i så fall bort kommentaren efter att koden genererats.

En annan del att belysa är också hur tidigare forskning behandlat promptande. Denny m.fl. (2022) har undersökt hur väl Copilot löser en uppgift när man ger Copilot själva uppgiftstexten och sen hur man behöver ändra den för att Copilot faktiskt skall lyckas lösa den. Det skiljer sig mot att faktiskt arbeta som systemutvecklare där man själv har till uppgift att lösa ett problem. I många fall ligger inte arbetet i att skriva koden utan snarare i att definiera problemet.

5.3 Komplex relation till större kod-stycken

Det råder en delad mening om att få stora stycken kod föreslagna av Copilot. Några av respondenterna upplever att stora stycken genererad kod är svåra att överblicka. Genom att dela upp och bara acceptera kortare stycken kan de lättare bibehålla kontroll över vad som skrivs och hur det skrivs. Samtidigt är det fler som efterfrågar större stycken kod. Några har då valt att komplettera med ChatGPT, men då är syftet inte alltid att använda koden direkt utan man ser det som inspiration eller en möjlig förklaring för att lösa uppgiften vilket stämmer överens med vad Bull & Kharrufa (2023) finner i sin studie. Några respondenter använder sig sedan av ChatGPT:s förslag för att skapa en egen lösning med hjälp av Copilot. Det arbetssättet tror vi framför allt är en ersättning till att använda sig av sökmotorer och forum för att söka lösningar på uppgifter. En stor del i när man väljer att acceptera större förslag från Copilot borde kunna kopplas till vilket sätt man just då arbetar på. Arbetar man *accelerande* och bara vill få ut lösningen man har i huvudet är det inte lika aktuellt att läsa igenom ett större, mer komplext förslag för att se om det stämmer med vad man tänkte, men arbetar man *undersökande* kanske större förslag kan bidra och inte störa i samma utsträckning.

5.4 ”With great power comes”... samma ansvar som alltid

Något som flera respondenter tar upp är vikten av att ta ansvar för sin kod. Oavsett om man skriver koden själv eller får den genererad så är det systemutvecklaren som står som ansvarig. För våra respondenter verkar *automation bias*, på det sätt som Al Madi (2022) och Chen m.fl. (2021) beskriver det, inte vara ett problem då det är en tydlig del i arbetet att kvalitetsgranska både för att lämna ifrån sig säker och underhållsbar kod.

Några har uttryckt en smärre oro att AI-generering av kod kan komma att göra systemutvecklare arbetslösa, men än så länge är det ingen som är övertygad om att så är fallet. Respondent 8 uttryckte det som att Copilot är ”*som en kompis på fyllan*”. Copilot kommer inte att ersätta en systemutvecklare, utan är snarare ett verktyg som gör systemutvecklare än vassare och samtidigt sänker ribban för mindre erfarna systemutvecklare att bidra. Vi kan också spekulera kring att Copilot kan öppna upp för personer med stark domänkunskap att bidra mer vid utvecklingsprojekt.

Det verkar finnas en viss oro på vissa arbetsplatser för att släppa in AI-verktyg i det vardagliga arbetet. Ovissheten kring ägandet av koden är en orsak, då källan till kunskapen i modellerna är tagen från Github. Även om källorna från Github är *open source*, fri källkod, så är man rädd för att testa då man inte vill vara först ut. En av respondenterna arbetar på en myndighet där användande av Copilot är hårt reglerat. Vad vi förstår handlar det om reglering kring att skicka data via servrar man inte har full kontroll över. Det är dock inget vi undersökt närmare.

Man kan argumentera för att ansvaret för koden redan är fördelat. I och med att man gör sig beroende av till exempel sorteringsfunktioner som kommer med högnivåspråk eller paket. Detta är ju redan kod som man inte utvecklat själv men litar på att det fungerar och därmed redan delegerat ansvar. Copilot jämförs på det här sättet med en högre nivå av språk av några respondenter, och i en framtid där verktyget har blivit bättre kan man kanske anta att delar av ansvaret läggs över mer på Copilot precis som vi nu ser med dagens högnivåspråk.

5.5 Parprogrammering: människa vs AI

Samtliga av respondenterna i undersökningen har varit överens om att Copilot inte fyller rollen som en mänsklig parprogrammerare. De största anledningarna till detta hör ihop med att deras syn på parprogrammering är inte huruvida man kan skriva snabbare kod, utan att det handlar om att komma fram till kod med bättre kvalitet samt att det skapar ett forum för att kunna diskutera och lösa komplexa problem genom att vara två hjärnor som samarbetar. Detta stärks också till stora delar av den tidigare forskning som finns gällande parprogrammering som menar att det nödvändigtvis inte går snabbare men att det blir lättare att lösa komplexa problem (Hannay m.fl., 2009; Imai, 2022). Det är ett område som respondenterna till viss del kompletterar med ChatGPT. ChatGPT har en resonerande förmåga och man kan på ett annat sätt diskutera fram lösningar genom att den bibehåller kontexten i den tidigare chatten, något som även framkommer av Bull & Kharrufa (2023). Den största skillnaden är att ChatGPT kan ta en ny prompt för att förändra sitt tidigare svar och är samtidigt medveten om vad den svarat tidigare. Respondenternas kombination av Copilot och ChatGPT liknar på många sätt parprogrammering så som både de själva och tidigare forskning beskriver (Nosek, 1998; Shiraishi m.fl., 2019; Williams m.fl., 2000).

Trots att alla respondenterna i undersökningen har varit överens om att Copilot inte fyller rollen som en parprogrammerare, har flera respondenter ändå identifierat delar av parprogrammering som kan liknas vid hur de använder Copilot. Bland annat har flera respondenter nämnt att det finns fördelar med att låta Copilot skriva kod och då i stället själva kunna lägga fokus på att granska koden då det kan vara lättare att vara objektiv när man granskar någon annans kod. Detta har också framkommit som en fördel med parprogrammering av flera respondenter samt tidigare forskning (Bird m.fl., 2022; Hannay m.fl., 2009; Nosek, 1998). Så trots att ingen av respondenterna direkt uttryckte detta, framgår det ändå att Copilot kan användas som en form av parprogrammerare men det blir tydligt att det bara berör begränsade delar i nuläget.

I en framtid där det är möjligt att kommunicera och diskutera med Copilot på en mer avancerad nivå, så kan det antas att Copilot också kan fylla dessa likheter med parprogrammering och vi kanske då kan prata om Copilot som en riktig och fullt utvecklade AI-parprogrammerare.

5.6 Kvalitet på genererad kod

Vi ser att kvaliteten på koden som genereras av Copilot är en stor inledande faktor i om man väljer att börja använda verktyget. Flera av respondenterna som testade Copilot under dess tidiga beta-period var inte nöjda med kvalitén och valde att inte fortsätta använda Copilot. Flera uppger att de valde att börja testa igen senare på grund av den rådande hajpen kring AI, framför allt kring ChatGPT. Upplevelsen var för alla betydligt mer tillfredställande. Däremot kan vi inte säkerställa om det faktiskt är kvalitén som blivit bättre eller om de har en annan toleransnivå till tekniken, alternativt att de blivit bättre på att arbeta med verktyget. Det är också några som uppskattade Copilot från start, till den grad att de uttryckt frustration under perioden mellan beta-testet och att deras företag köpt in Copilot. De hade då inte tillgång till verktyget som de vant sig vid, och kände att detta hämmade deras arbete.

Något som är viktigt för respondenterna är att koden de skriver blir lätt att förstå, och därmed att underhålla för kommande systemutvecklare. Det påverkar deras val att acceptera de förslagen som ges av Copilot när det används som *autocomplete*. För de flesta utgör det inget hinder att Copilot ger undermåliga förslag, de skriver bara vidare tills förslaget är bra nog.

Flera respondenter har också nämnt att de tycker att koden som Copilot skriver oftast är välskriven och lättläst vilket både underlättar granskningen av koden och fyller ett syfte att kunna förstå koden i framtiden. Att koden som Copilot skriver har en hög läsbarhet är också något som tagits upp av Nguyen & Nadi (2022) i deras studie. Vikten av att koden är läsbar blir tydligt när man ser till att majoriteten av respondenterna föredrar att skriva läsbar kod framför effektiv kod i syfte av att koden ska vara självförklarande och slippa skriva kommentarer. Sättet som Finnie-Ansley m.fl.

(2022, 2023) bedömer Copilots förmåga att lösa problem på blir då ganska fränkopp-
lad från hur majoriteten av respondenterna arbetar. Hur stor del av Copilots kod som
är direkt användbar är alltså inte speciellt viktigt, då de snarare vill att den ska se ut
på ett visst sätt. Flera respondenter nämner också att även om förslagen från Copilot
direkt inte löser problemet så går det ofta att lätt redigera koden tills att den funkar.
Att koden har en hög läsbarhet kan antas vara en anledning till att det därför är lättare
att förstå och redigera de eventuella fel som krävs för att lösa problemet vilket gör
detta arbetssätt möjligt.

6 Reflektion

En stor poäng att åter lyfta är att i en kvalitativ explorativ studie är resultatet inte generaliserbart. Hade andra respondenter medverkat i studien, så hade resultatet förmodligen sett annorlunda ut. Syftet med studien är heller inte att påvisa en generaliserbar slutsats, utan endast att lyfta erfarenheter från verkligt användande och ställa dessa mot tidigare forskning. Med det i åtanke hade vi tyckt det vore intressant att intervjua personer med lite mer olika relationer till programmering, såsom systemutvecklare med väldigt höga krav på optimering av kod eller systemutvecklare som har ett mer konstnärligt förhållande till kod. Båda synsätten har dykt upp under intervjuerna, beskrivet av respondenter som potentiellt problematiska kring användandet av Copilot.

Av de respondenter vi fick tag på så har samtliga varit positivt inställda till att använda Copilot. Det hade därför varit intressant att hitta personer som varit mer negativt inställda efter att de använt Copilot vilket kunde ha gett mer nyans åt begränsningar och eventuella risker.

Då det endast gjordes en avgränsning till att respondenterna skulle ha använt verktyget och inte till hur länge, så gick det att urskilja att de respondenter som endast använt verktyget under en kortare period gav mer spekulerande svar på en del intervjufrågor. Att sätta en minimigräns på användandet kunde därför varit givande. Denna avgränsning skulle dock kunna krocka med ovan resonemang att man ville få in fler som varit negativt inställda till att använda Copilot, vilka då högst troligt inte använt verktyget under en längre period.

Det blir också tydligt under intervjuerna att respondenterna inte alltid själva är helt medvetna över hur och varför de arbetar med verktyget på ett visst sätt, utan det krävdes oftast en del grävande av intervjuaren. Det hade därför varit intressant att utföra observationer när respondenterna jobbar för att ytterligare berika studien. Vidare hade en kvantitativ studie av hela populationen med underlag av denna studie kunna berika och svara på frågan om generaliserbarhet.

7 Slutsats

Det är tydligt att respondenterna vi intervjuat är nöjda med vad de får ut av Copilot. Användandet av Copilot verkar till stor del komma från tidigare arbetssätt. De flesta respondenterna använder Copilot enbart som *autocomplete*, och endast en arbetar aktivt med kommentarer i naturligt språk. Det verkar till stor del bero på deras inställning till att kommentarer i kod är onödigt och därför känns detta arbetssätt onaturligt. Det är tydligt att de flesta inte använder Copilot till sin fulla potential och någon form av utbildning skulle kunna höja effekten de får ut vid användande.

Det finns flera anledningar till att respondenterna använder Copilot. Inledningsvis handlar det om nyfikenhet till ny teknik och i flera fall den rådande hajp kring AI-hjälpmiddel. När de väl börjat använda Copilot upplever de en stor nytta i kodskrivande som anses tråkigt, och på så sätt minska kognitiv belastning. Användandet minskar informationssökande i dokumentation och skyndar därmed på arbetet. Copilot utmanar deras tidigare sätt att skriva kod och de kan på så sätt utvecklas som programmerare. Det kan antas att anledningen till att respondenterna upplever en stor vinning med Copilot hänger ihop med deras inställning till programmering. De ser sig i många fall som produktutvecklare och programmering är då endast ett verktyg. Copilot effektiviserar det verktyget så de kan lägga mer tid på uppgifter de anser vara mer meningsfulla. Eventuellt skulle det kunna vara så att personer som har en annan syn på programmering inte upplever samma positiva effekt.

Några liknar Copilot med en ny, högre nivå av programmeringsspråk, och hur det sänker ribban för fler att börja programmera. Då skulle man kunna tänka sig att andra yrkesgrupper med djupare domänkunskap kan vara med och bidra till projekten.

Copilot lever enligt respondenterna inte upp till att vara en parprogrammerare. Vissa likheter tyckes finnas, så som att man tar en navigerande roll och granskar koden, men vad som framför allt lyfts i intervjuerna är att parprogrammering handlar om att vara två eller fler som diskuterar och bollar idéer. Där kan i dagsläget inte Copilot bidra, och då har flera i stället vänt sig till ChatGPT som kan bidra som bollplank. I den aspekten har Copilot möjlighet att utvecklas i hur man interagerar med verktyget.

Det verkar onekligen som att Copilot är ett kraftfullt och värdefullt verktyg även för professionella systemutvecklare.

8 Referenser

- Advent of Code*. (u.å.). Hämtad 17 maj 2023, från <https://adventofcode.com/2022/about>
- Al Madi, N. (2022). *How Readable is Model-generated Code? Examining Readability and Visual Inspection of GitHub Copilot*. ACM International Conference Proceeding Series. Scopus. <https://doi.org/10.1145/3551349.3560438>
- ATLAS.ti*. (u.å.). ATLAS.Ti. Hämtad 28 april 2023, från <https://atlasti.com>
- Barke, S., James, M. B., & Polikarpova, N. (2022). *Grounded Copilot: How Programmers Interact with Code-Generating Models* (arXiv:2206.15000). arXiv. <https://doi.org/10.48550/arXiv.2206.15000>
- Beck, K. (2000). *Extreme Programming Explained: Embrace Change*.
- Bell, J. (2016). *Introduktion till Forskningsmetodik* (B. Nillson, Övers.; 5:2). Studentlitteratur.
- Bird, C., Ford, D., Zimmermann, T., Forsgren, N., Kalliamvakou, E., Lowdermilk, T., & Gazit, I. (2022). *Taking Flight with Copilot*. <https://doi.org/10.1145/3582083>
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101. <https://doi.org/10.1191/1478088706qp063oa>
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). *Language Models are Few-Shot Learners* (arXiv:2005.14165). arXiv. <http://arxiv.org/abs/2005.14165>

- Bull, C., & Kharrufa, A. (2023). *Generative AI Assistants in Software Development Education* (arXiv:2303.13936). arXiv. <https://doi.org/10.48550/arXiv.2303.13936>
- Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. de O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code* (arXiv:2107.03374). arXiv. <https://doi.org/10.48550/arXiv.2107.03374>
- Computer History: A Timeline of Computer Programming Languages | HP® Tech Takes*. (u.å.). Hämtad 09 maj 2023, från <https://www.hp.com/us-en/shop/tech-takes/computer-history-programming-languages>
- Dakhel, A. M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., Ming, Z., & Jiang. (2022). *GitHub Copilot AI pair programmer: Asset or Liability?* (arXiv:2206.15331). arXiv. <https://doi.org/10.48550/arXiv.2206.15331>
- Denny, P., Kumar, V., & Giacaman, N. (2022). *Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language* (arXiv:2210.15157). arXiv. <https://doi.org/10.48550/arXiv.2210.15157>
- Ethical Guidelines for Online Interviews | Virtual Ethnographic Methods | Class Research Portfolio*. (u.å.). Hämtad 23 mars 2023, från <https://voices.uchicago.edu/202003sosc20224/2020/06/25/ethical-guidelines-for-online-interviews/>
- Finnie-Ansley, J., Denny, P., Becker, B. A., Luxton-Reilly, A., & Prather, J. (2022). *The robots are coming: Exploring the implications of OpenAI codex on introductory programming*. 10–19. Scopus. <https://doi.org/10.1145/3511861.3511863>
- Finnie-Ansley, J., Denny, P., Luxton-Reilly, A., Santos, E. A., Prather, J., & Becker, B. A. (2023). *My AI Wants to Know if This Will Be on the Exam: Testing*

- OpenAI's Codex on CS2 Programming Exercises*. 97–104. Scopus. <https://doi.org/10.1145/3576123.3576134>
- GitHub Copilot · Your AI pair programmer*. (u.å.). GitHub. Hämtad 13 februari 2023, från <https://github.com/features/copilot>
- Hannay, J., Dybå, T., Arisholm, E., & Sjøberg, D. (2009). The effectiveness of pair programming: A meta-analysis. *Information and Software Technology*, 51, 1110–1122. <https://doi.org/10.1016/j.infsof.2009.02.001>
- Imai, S. (2022). *Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study*. 319–321. Scopus. <https://doi.org/10.1109/ICSE-Companion55297.2022.9793778>
- IntelliSense in Visual Studio Code*. (u.å.). Hämtad 06 mars 2023, från <https://code.visualstudio.com/docs/editor/intellisense>
- Introducing ChatGPT*. (u.å.). Hämtad 03 maj 2023, från <https://openai.com/blog/chatgpt>
- Introducing Whisper*. (u.å.). Hämtad 28 april 2023, från <https://openai.com/research/whisper>
- Jaworski, M., & Piotrkowski, D. (2023). *Study of software developers' experience using the Github Copilot Tool in the software development process* (arXiv:2301.04991). arXiv. <https://doi.org/10.48550/arXiv.2301.04991>
- Jiang, E., Toh, E., Molina, A., Olson, K., Kayacik, C., Donsbach, A., Cai, C. J., & Terry, M. (2022). Discovering the Syntax and Strategies of Natural Language Programming with Generative Language Models. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–19. <https://doi.org/10.1145/3491102.3501870>
- Kalliamvakou, E. (2022, september 7). Research: Quantifying GitHub Copilot's impact on developer productivity and happiness. *The GitHub Blog*.

<https://github.blog/2022-09-07-research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>

- Mozannar, H., Bansal, G., Fourney, A., & Horvitz, E. (2022). *Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming* (arXiv:2210.14306). arXiv. <https://doi.org/10.48550/arXiv.2210.14306>
- Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and Organization*, 17(1), 2–26. <https://doi.org/10.1016/j.infoandorg.2006.11.001>
- Nguyen, N., & Nadi, S. (2022). *An Empirical Evaluation of GitHub Copilot's Code Suggestions*. 1–5. Scopus. <https://doi.org/10.1145/3524842.3528470>
- Nosek, J. (1998). The Case for Collaborative Programming. *Communications of the ACM*, 41. <https://doi.org/10.1145/272287.272333>
- OpenAI Announces 12 Billion Parameter Code-Generation AI Codex*. (u.å.). InfoQ. Hämtad 08 maj 2023, från <https://www.infoq.com/news/2021/08/openai-codex/>
- Patel, R., & Davidsson, B. (2019). *Forskningsmetodikens grunder* (5:e uppl.). Studentlitteratur.
- Professionell | SAOB*. (u.å.). Hämtad 29 mars 2023, från https://www.saob.se/artikel/?unik=P_1843-0221.32U2
- Sarkar, A., Gordon, A. D., Negreanu, C., Poelitz, C., Ragavan, S. S., & Zorn, B. (2022). *What is it like to program with artificial intelligence?* (arXiv:2208.06213). arXiv. <https://doi.org/10.48550/arXiv.2208.06213>
- Shiraishi, M., Washizaki, H., Fukazawa, Y., & Yoder, J. (2019). Mob Programming: A Systematic Literature Review. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, 2, 616–621. <https://doi.org/10.1109/COMPSAC.2019.10276>

- Systemutvecklare*. (u.å.). Unionen. Hämtad 16 maj 2023, från <https://www.unionen.se/rad-och-stod/om-lon/marknadsloer/systemutvecklare>
- Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). *Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models*. Conference on Human Factors in Computing Systems - Proceedings. Scopus. <https://doi.org/10.1145/3491101.3519665>
- Waldinger, R. J., & Lee, R. C. T. (1969). PROW: A step toward automatic program writing. *Proceedings of the 1st international joint conference on Artificial intelligence*, 241–252.
- Walsham, G. (1995). Interpretive case studies in IS research: Nature and method. *European Journal of Information Systems*, 4(2), 74–81. <https://doi.org/10.1057/ejis.1995.9>
- Welsh, M. (2022). The End of Programming. *Communications of the ACM*, 66(1), 34–35. <https://doi.org/10.1145/3570220>
- Wermelinger, M. (2023). *Using GitHub Copilot to Solve Simple Programming Problems. 1*, 172–178. Scopus. <https://doi.org/10.1145/3545945.3569830>
- Williams, L., 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>

9 Bilagor

Bilaga 1 - Intervjuguide

Inledning

- Presentera studien kortfattat.
- Presentera konfidentialitet
- Presentera lagring av data
- Stäm av om det är okej att spela in

Bakgrund

Hur länge har du arbetat som programmerare?

- Professionellt
- Privat

Beskriv kort din arbetsroll.

- Hur mycket kodar du på en dag?
- Frontend, backend?
- Jobbar du i team?
- Ledande roll?
- Hur ser din arbetsdag ut?

Vad programmerar du i för språk primärt? (Vilken typ av kod skriver du?)

Vad är programmering för dig?

Roligaste/tråkigaste med att skriva kod?

- Vad upplever du är den största utmaningen?

Vad upplever du är den största utmaningen?

Användande

Varför valde du att börja använda Copilot?

Använder du Copilot idag?

Ja

- Hur använder du Copilot?
- Hur kom du fram till det?
- Hur länge har du använt Copilot?
- Beskriv hur du går till väga
- Hjälper det dig?

När fungerar det bra?

När fungerar det inte?

Nej

Varför inte?

Hur använde du Copilot?

- Hur länge har du använt Copilot?
- Beskriv hur du gick till väga
- Hjälpte det dig?

När fungerar det inte?

När fungerar det bra?

Personlig åsikt

Vad har du för åsikt gällande Copilot?

- Ser du några andra fördelar med att använda Copilot?
- Ser du några andra nackdelar med att använda Copilot?

Copilot marknadsförs som en AI-par-programmerare. Hur ser du på det?

- Har du jobbat med parprogrammering nu/innan?

Saknar du något med Copilot?

Områden som framkommit i tidigare studier samt intervjuer

- Accelererande
- Utforskande
- Autocomplete
- Promptar
- Naturligt språk
- Säkerhet
- Mockdata?
- Tester
- ChatGPT

Bilaga 2 – Koder Atlas.ti

● ◆ Ansvar för kod man producerar	37
● ◆ Användande - Fungerar bra	23
● ◆ Användning - Accelererande	35
● ◆ Användning - Accelererande/Utforskande	34
● ◆ Användning - Anpassat sitt arbetssätt efter Copilot	42
● ◆ Användning - Fungerar inte bra	28
● ◆ Användning - Hur	95
● ◆ Användning - Tid	13
● ◆ Användning - Till	26
● ◆ Användning - Utforskande	20
● ◆ Användning - Valt bort vissa delar	10
● ◆ Användning - Varför	21
● ◆ Bakgrund	67
● ◆ Bakgrund - Arbetsroll/arbetsuppgifter	35
● ◆ Bakgrund - Kodkommentarer	2
● ◆ Bakgrund - Relation till programmering	54
● ◆ Bakgrund - Roligt	28
● ◆ Bakgrund - Tråkigt	17
● ◆ Bra citat	21
● ◆ ChatGPT	36
● ◆ Nyttor	61
● ◆ Nyttor - Fokus/Flow	8
● ◆ Nyttor - Frigör kognitiv kapacitet	10
● ◆ Nyttor - Lärande	11
● ◆ Nyttor - Minskar repetitiva uppgifter	11
● ◆ Nyttor - Slipper googla/läsa documentation	14
● ◆ Parprogrammering	36
● ◆ Risker med att använda Copilot	23
● ◆ Åsikt - Fördelar	21
● ◆ Åsikt - Inställning till Copilot	39
● ◆ Åsikt - Nackdelar	23
● ◆ Åsikt - Vad saknas i Copilot/ Copilots framtid	41
● ◆ Åsikt - Vad tycker andra om Copilot?	14

Bilaga 3 – Mall mailutskick

Hej XXX

Vi är en grupp studenter som skriver examensarbete på Institutionen för tillämpad IT, Göteborg universitet där vi undersöker utvecklarens erfarenheter av Github Copilot. Vi letar efter erfarna systemutvecklare (4+ år) som tidigare testat Copilot eller är användare. Studiens syfte är att undersöka faktorer som påverkar valet att använda Copilot. Skulle du kunna tänka dig att bli intervjuad kring din erfarenhet av Copilot? Intervjun beräknas ta ca 45 min men vi är flexibla.

Du kommer att vara anonym och det är helt okej att när som helst välja att dra dig ur. Intervjuerna kommer primärt att genomföras under vecka 14, 15 och 16.

Om du är intresserad, så svara på det här mailet så bokar vi in en tid och plats som passar dig. Om du har frågor är det bara att höra av sig via mail eller telefon.

Vi är även intresserade av att nå ut till fler personer. Om du känner till någon som du tror skulle vara intresserad av att ställa upp i vår studie är vi väldigt tacksamma om du kan förmedla den kontakten vidare till oss. Det gäller både aktiva användare och systemutvecklare som testat men inte använder längre.

Bästa Hälsningar

Gustaf, August och Mårten