LARGE LANGUAGE MODELS

Bedömning av ChatGPT:s potential som verktyg för kommentering av kod

Kandidatuppsats i Informatik

Dennis Vuk
Tom Svensson

2023KANI32

HÖGSKOLAN I BORÅS
INSTITUTIONEN HANDELS- OCH IT-HÖGSKOLAN

Svensk Titel: Large Language Models: Bedömning av ChatGPT:s potential som verktyg för kommentering av kod

Engelsk Titel: Large Language Models: Assessment of ChatGPT's Potential as a Tool for Code Commenting

Utgivningsår: 2023

Författare: Dennis Vuk & Tom Svensson

Handledare: Rikard König

Abstract

The usage of Artificial Intelligence (AI) is widespread among both companies and individuals today. It has become an integrated part of our society, often going unnoticed. From face recognition and self-driving cars to automation in work-related areas, AI has undeniably impacted the world. As AI models continue to evolve, concerns about their impact on jobs, associated security risks, and ethical dilemmas arise. The literature in this essay helps portray AI historically, in the present, and provides an insight into its future direction. The AI model that has currently garnered the most attention is ChatGPT. Its potential seems limitless, which prompted the relevance of increasing knowledge about the AI model. Furthermore, a delimitation was made, where the focus area was to investigate how ChatGPT can generate code comments and potentially act as a tool for commenting source code. As part of the research focus and scope, the research question was formulated: "Large Language Models: Assessment of ChatGPT's Potential as a Tool for Code Commenting."

To answer the research question, the thesis adopted a qualitative approach, with programmers as the selected respondents. The primary data collection was conducted through two semi-structured interviews, where the initial interview involved capturing initial impressions of ChatGPT and gathering general information about the interviewees. Additionally, an observation was carried out to gain insights into how programmers utilize the AI model, followed by a post-observation interview to gather the interviewees' thoughts after using ChatGPT to generate code comments.

Based on the collected empirical data, the study was able to conclude certain limitations in the current model, particularly the need for clear instructions. Despite these limitations, ChatGPT's performance demonstrates the potential to be a significant resource for code commenting in the future. The results indicate that the model can generate relatively suitable comments in the analyzed code snippets. However, during the concluding interviews, participants generally expressed that the comments were redundant and lacked significant value in enhancing the understanding of the source code. Nevertheless, the respondents

discussed the possibilities of using ChatGPT in the future, while emphasizing the need for improvements to establish it as a reliable method in work-related situations.

Keywords: Artificial Intelligence, Large Language Models (LLM), ChatGPT, OpenAI, Code, Comments, Programmer, Software Developer, Documentation

Sammanfattning

Användningen av Artificiell Intelligens (AI) är utbredd bland verksamma företag idag, likväl privatpersoner. Det har blivit en integrerad del av vårt samhälle som ofta går obemärkt förbi. Allt från face recognition, självkörande bilar och automatisering inom arbetsrelaterade områden, har AI onekligen påverkat omvärlden. I takt med att AI-modeller fortsätter att utvecklas tillkommer även farhågor om dess påverkan på jobb, tillhörande säkerhetsrisker och etiska dilemman. Uppsatsens litteratur hjälper till att skildra AI historiskt, i nutid, men även ge en uppfattning om vart den är på väg. Den AI-modell som i nuläget har väckt störst uppmärksamhet är ChatGPT. Dess potential tycks inte ha några gränser, därmed uppstod relevansen för att öka kunskapen kring AI-modellen. Vidare gjordes en avgränsning, där fokusområdet var att undersöka hur ChatGPT kan generera kodkommentarer och potentiellt agera som ett hjälpmedel vid kommentering av källkod. I samband med avgränsningen och fokusområdet bildades även forskningsfrågan: Large Language Models: Bedömning av ChatGPT:s potential som verktyg för kommentering av kod

För att besvara forskningsfrågan har avhandlingen varit baserat på en kvalitativ ansats, där urvalet av respondenter har varit programmerare. Den primära datainsamlingen har genomförts via två semistrukturerade intervjuer, varav den inledande innefattade initiala känslor kring ChatGPT och övergripande fakta om respektive intervjuobjekt. Vidare gjordes det en observation för att få en inblick i hur AI-modellen används av programmerare, för att avslutningsvis göra en uppföljande intervju post-observation i syfte att samla tankarna från intervjuobjekten efter användning av ChatGPT för att generera kodkommentarer.

Baserat på den insamlade empirin kunde studien konkludera vissa begränsningar i den nuvarande modellen, inte minst behovet av tydliga instruktioner. Trots brister visar ChatGPTs framställning potential att vara en betydande resurs för kommentering av kod i framtiden. Resultaten indikerar att modellen kan generera relativt passande kommentarer i de analyserade kodkodstycken. Emellertid uttryckte deltagarna under de avslutande intervjuerna generellt sett att kommentarerna var redundanta och saknade betydande värde för att öka förståelsen av källkoden. Respondenterna diskuterade dock möjligheterna att använda ChatGPT i framtiden, men underströk behovet av förbättringar för att göra det till en tillförlitlig metod inom arbetsrelaterade situationer.

Nyckelord: Artificiell Intelligens (AI), Large Language Models (LLM), ChatGPT, OpenAI, Kod, Kommentarer, Programmerare, Mjukvaruutvecklare, Dokumentation

Förord

Inledningsvis vill vi rikta ett stort tack till vår handledare Rikard König, som med sin bakgrund inom IT och i synnerhet artificiell intelligens kunde vägleda oss med sina kloka råd. Vi vill även tacka samtliga respondenter som tog sig tid att vara delaktiga i två intervjuer och en observation. Avslutningsvis vill vi också tacka Dr. Gideon Mbiydzenyuy och övriga opponenter för konstruktiv feedback som gav upphov till att vi kunde färdigställa uppsatsen. Utan er hade studien inte varit möjlig.

Trevlig läsning!
Dennis Vuk & Tom Svensson

Innehållsförteckning

1.Introduktion **6**
1.1 Bakgrund 6
1.2 Forskningsöversikt 7
1.3 Problemdiskussion 9
1.4 Problemformulering 9
1.5 Syfte och forskningsfråga 10
1.6 Förväntat resultat 10
1.7 Målgrupp 10
1.9 Disposition 11
2. Metod **12**
2.1 Forskningsstrategi 12
2.2 Urval av respondenter 13
2.3 Datainsamling 13
2.3.1 Litteraturstudier 14
2.3.2 Intervjuer 14
2.3.3 Observationer 14
2.4 Analys av empirisk data 15
2.5 Metodreflektion 15
2.5.1 Reliabilitet 17
2.5.2 Validitet 17
2.5.3 Generaliserbarhet 17
2.5.4 Konsekvenser av avgränsning 18
3. Teoretiskt ramverk **19**
3.1 ChatGPT, dess bakgrund och funktion 19
3.2 Kommentering av kod 20
3.2.1 Kodkonventioner 21
3.2.2 Python 22
3.2.3 PHP 22
3.2.4 JavaScript 22
3.3 Konsekvenser vid bristande kommentering av kod 23
3.4 Popularitet bland programmeringsspråk 24
3.5 Utmaningar med ChatGPT 25
3.6 ChatGPT som hjälpmedel inom programmering 25
3.7 Etiska dilemman, lagar och regler 26
4. Empiri/Resultat **27**
4.1 Inledande intervju 27
4.1.1 Respondent A 27
4.1.2 Respondent B 28
4.1.3 Respondent C 29
4.1.4 Respondent D 30

4.2 Observationer 31
4.2.1 Respondent A (PHP) 33
4.2.2 Respondent B (ERLANG) 35
4.2.3 Respondent C (Javascript) 37
4.2.4 Respondent D (Python) 39
4.3 Avslutande intervjuer 41
4.3.1 Respondent A 41
4.3.2 Respondent B 42
4.3.3 Respondent C 43
4.3.4 Respondent D 43
5. Analys 45
5.1 Inledande intervju 45
5.1.1 Tjänsteställning/erfarenheter 45
5.1.2 Riktlinjer vid kommentering av kod 46
5.1.3 Hjälpmedel vid kommentering av kod 47
5.1.4 Innebörd av att kommentera kod 48
5.1.5 Komplikationer och brister vid kommentering 49
5.1.6 Large language models 51
5.2 Observation 52
5.3 Avslutande intervju 53
5.3.1 Respondent A (PHP) 53
5.3.2 Respondent B (ERLANG) 54
5.3.3 Respondent C (JavaScript) 54
5.3.4 Respondent D (Python) 56
6. Diskussion och slutsatser 57
6.1 Resultatdiskussion 57
6.2 Vidare forskning 58
7. Källförteckning 60
8. Bilagor 63
Intervjuguide Inledande Frågor 63
Intervjuguide Avslutande frågor 63
Bilaga Observationer 64
JAVASCRIPT (FULL CODE, okommenterad) 66
JAVASCRIPT (FULL CODE, kommenterad) 78
Erlang (FULL Code, okommenterad) 100
Erlang (FULL code, kommenterad) 102

1.Introduktion

Detta avsnitt innefattar en bakgrund som presentation för valt ämne till uppsatsarbetet, som sedan följs upp av tidigare studier i form av forskningsöversikt. Vidare tydliggörs studiens syfte och forskningsfråga.

1.1 Bakgrund

Vi befinner oss i mitten av en digital era som på relativt kort tid inneburit många förändringar för samhället vi känner till idag. På senare år har det tillkommit flera tekniska innovationer som haft en stor inverkan på våra liv. Artificiell Intelligens (AI) är ett tydligt exempel på en teknologi som influerat oss människor inom ett flertal olika områden. AI är en generell term och beskrivs enligt Haenlein & Kaplan (2019, s. 5) som *"a system's ability to interpret external data correctly, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation"*. Idag kan vi se många olika typer av AI som används dagligen och ofta går obemärkt förbi. Makridakis (2017) skriver om flertalet exempel i sin artikel *The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms*. Författaren nämner bland annat att AI hjälper användare av smartphones att komplettera ord som inte är färdigskrivna, att det nuförtiden finns självkörande bilar som navigerar via AI och att GPS funktionen i smartphones samt surfplattor förser användaren med den mest lämpliga färdsträckan till önskvärd destination (Makridakis, 2017).

Programmering är en grundpelare inom AI och är även det som gett upphov till AIs existens. Framträdandet av OpenAIs Large Language Model (LLM) ChatGPT, kommer eventuellt skapa nya utmaningar för mjukvaruutvecklare inom en snar framtid (Ross et al, 2023). Det finns många möjligheter i samband med introduktionen av ChatGPT, men det finns även konsekvenser som ännu inte har undersökts. På så vis är det intressant att få en inblick inom båda delarna av spektrumet och ställa möjligheter kontra utmaningar mot varandra.Vidare uppstår intresset att undersöka ifall nuvarande teknologier kan automatisera vissa aspekter inom programmering, vilket eventuellt kan ge indikationer inför framtiden.

"DRY" är en akronym för "Don't repeat yourself". Det i sin tur klassas som den gyllene regeln inom programmering och är av stor vikt vid dokumentation av kod. Källkod och kodkommentarer hör ihop, då kommentarerna fungerar likt ett hjälpmedel för programmeraren att förstå vad ett specifikt kodstycke gör. Spinellis (2010) nämner att det i nuläget finns flera brister inom dokumentation av kod. Bland de främsta punkterna som framkommer är inkonsekvent användning av koddokumentation. Utvecklare har delade syner på vad som bör kommenteras och hur man ska gå tillväga. Således saknas det ett generellt tillvägagångssätt, vilket orsakar förvirring och svårigheter i att förstå kodbasen. Ytterligare problematik inom koddokumentation är onödiga kommentarer. Spinellis (2010) menar att slarvigt skrivna kommentarer är värre än kommentarer som saknas helt. Detta då det kan röra

om koden, göra det svårare att läsa och underhålla. Konsekvenserna blir därmed onödigt omarbete och större utgifter (Spinellis, 2010).

1.2 Forskningsöversikt

Large Language Models (LLMs) är AI-modeller som via stora mängder textdata lär sig att förstå och skapa mänskligt språk. De är en sammanställning av neurala nätverk och maskininlärningsalgoritmer, i syfte att bearbeta språk på ett snarlikt sätt som människor gör. LLMs används idag inom ett brett spektrum av tillämpningar. Det inkluderar bland annat språköversättningar, chatbots och textsammanfattning. ChatGPT har visat sig vara användbart i förhållande till olika arbetsuppgifter inom programmering. AI-modellen kan bland annat generera kodavsnitt baserat på användarinmatning, verka som ett hjälpmedel inom dokumentation och felsöka problem mer tidseffektivt (Ray, 2023).

Förvaltning av system är en tidskrävande och dyr process inom systemutveckling. Det har framförts flera förslag på tillvägagångssätt för att förbättra förståelsen av källkod, sedermera för att minska förvaltningskostnader. Användning av kommentarer är ett exempel på tillvägagångssätt som tillåter utvecklare att beskriva källkod med egna ord. I en studie genomförd av Sebastian Nielebock et al (2019) undersöks 277 personer, främst med expertis och mångårig erfarenhet inom systemutveckling. Resultatet visade på att majoriteten av respondenterna som använde kommentarer i sin kod producerade lösningar som var lättare att förstå och även att ändra (Nielebock et al, 2019). Kommentering av kod är enligt Huang et al (2019) & Nielebock et al (2019) en viktig men ofta förbisedd aktivitet inom systemutveckling. Det finns även vissa brister för formella och automatiska tillvägagångssätt i dagens praxis, just i avseende på kodkonventioner (Huang et al, 2019).

I stora projekt inom mjukvaruutveckling, där flera utvecklare samarbetar under långa perioder, uppstår utmaningar när källkod inte följer etablerade konventioner. Detta bekräftas av Laramee (2013), som framhåller att skrivandet av kod utan att följa konventioner leder till svårigheter att återanvända kod, vilket i sin tur resulterar i att utvecklare behöver återuppföra liknande lösningar gång på gång. En annan utmaning inom programmering är att kommentarer i källkoden ofta förbises av professionella programmerare, vilket påpekas av Salviulo & Scanniello (2014). Denna tendens kan leda till att koden blir svårtolkad, även för den ursprungliga utvecklaren. Vidare konstaterar Laramee (2013) att bristen på, eller föråldrade kommentarer skapar komplikationer när utvecklare behöver uppdatera befintlig kod och har svårigheter att sätta sig in i källkodens ursprungliga struktur.

Kommentarer i källkod spelar en avgörande roll som ett sätt att dokumentera själva koden. Enligt Wen et al (2019) är det dock en tidskrävande process som kräver högt fokus att hålla kommentarerna uppdaterade i takt med ändringar i källkoden. I studien *"A Large-Scale Empirical Study on Code-Comment Inconsistencies"* undersöker skribenterna hur källkoden och dess tillhörande kommentarer utvecklas tillsammans. När utvecklare utför tester, åtgärdar

buggar eller omstrukturerar en klass behöver de först ha kunskap om de berörda kodkomponenterna. Studien presenterar också tidigare forskning som visar att cirka 60% av tiden som mjukvaruutvecklare spenderar i källkoden går åt till att förstå dess struktur, innebörd och sammanhang. Forskning genomförd av Nielebock et al (2019) visar också att källkod med tydliga och välkommenterade kommentarer är lättare att förstå än källkod utan kommentarer. Trots detta är det sällsynt att mjukvaruutvecklare ges möjlighet att noggrant kommentera ny kod eller att uppdatera befintliga kommentarer efter kodändringar. Enligt Nielebock et al (2019) kan det bero på tidsbrist och låg motivation. Om en metodkommentar inte uppdateras efter stora förändringar i metodens applikationslogik kan kommentaren ge felaktig information till utvecklare som försöker förstå metoden. Detta kan i sin tur försvåra programförståelsen istället för att underlätta den. Wen et al (2019) betonar vikten av att investera ansträngning i utvecklingen av verktyg som stödjer omarbetning och omstrukturering av kodkommentarer (Wen et al, 2019).

I tillägg till dessa utmaningar kan projektet också påverkas av faktorer som tidsbegränsningar och bristande motivation, vilket Nielebock et al (2019) framhäver. Forskning visar emellertid att kodkonventioner, inklusive kommentarer, utgör en viktig grund för framgångsrik mjukvaruutveckling. Enligt Laramee (2013) kan följsamma konventioner underlätta för olika individer, både inom och utanför utvecklingsteamet, att förstå och hantera källkoden. Salviulo & Scanniello (2014) betonar att kommentarer i källkoden är särskilt värdefulla för nybörjare, då de bidrar till ökad förståelse för olika program. Dessutom lyfter Smit et al (2011) fram betydelsen av en tydligt definierad namngivningsstil som överensstämmer med befintliga bibliotek, vilket underlättar kopplingen mellan syntax och semantisk betydelse. Välskrivna kommentarer är enligt Smit et al (2011) till stor hjälp för nya utvecklare att förstå befintlig källkod samt för att främja enhetlighet inom organisationer och team, samt underlätta användningen av olika API.

1.3 Problemdiskussion

Denna uppsats syftar till att undersöka användningen av Large Language Models (LLMs) och dess potential för att automatisera kodkommentarer inom mjukvaruutveckling. För att få en uppfattning om värdet av automatisering av kodkommentarer kommer studiens fokus att riktas mot erfarna programmerare och deras åsikter kring ChatGPT som hjälpmedel inom koddokumentation. Tidigare forskning pekar på att kommentarer i källkod är avgörande för att dokumentera och underlätta förståelsen av källkod, men det finns enligt Wen et al (2019) utmaningar med att hålla kommentarer uppdaterade och relevanta i takt med ändringar i källkoden. En av de utmaningar som LLMs kan adressera är den tidskrävande processen att skriva och uppdatera kodkommentarer. LLMs använder stora mängder textdata för att lära sig att förstå och generera mänskligt språk (Ray, 2023). Detta kan eventuellt underlätta skapandet av relevanta och korrekta kodkommentarer. Automatiserade kodkommentarer kan bidra till att minska arbetsbördan för utvecklare och frigöra tid för mer kritiska uppgifter, då mjukvaruutvecklare spenderar ungefär 60 % av tiden för att förstå källkod (Wen et al, 2019). Genom att använda LLMs kan kodkommentarer genereras baserat på källkoden och dess sammanhang, vilket kan underlätta förståelsen och därmed underhåll av mjukvarusystem. Samtidigt kan andra problem inom koddokumentation adresseras, inte minst då det ofta råder tidsbrist och låg motivation (Nielebock et al, 2019). Erfarna mjukvaruutvecklare har även tendens att förbise kommentarer i kod, vilket försvårar för juniora utvecklare (Salviulo & Scanniello (2014). Genom att utföra en empirisk studie kan effekterna av automatiserade kodkommentarer undersökas. Det blir också intressant att undersöka om automatiserade kodkommentarer genererade av LLMs kan leverera liknande kvalitet och användbarhet som manuellt skrivna kommentarer och hur detta uppfattas av erfarna mjukvaruutvecklare.

1.4 Problemformulering

Valet av problemområde baserades på tidigare forskning inom dokumentation för källkod och dess innebörd för mjukvaruutvecklare. Samtidigt konstateras möjligheter för automatisering i takt med nya teknologier, som ChatGPT. Tidigare forskning pekar på att kommentarer är avgörande för att förbättra förståelse av källkod inom mjukvaruutveckling. Kommentering av kod och uppdateringen av kodkommentarer är dock en tidskrävande process som kräver fokus och engagemang från utvecklare. Genom att använda LLMs kan det finnas potential att automatisera genereringen av kodkommentarer baserat på källkoden och dess sammanhang.

1.5 Syfte och forskningsfråga

Studiens syfte är att undersöka huruvida Large Language Models kan hjälpa programmerare att kommentera källkod. Vidare kommer studiens respondenter att analysera och värdera ChatGPT i förhållande till kommentering av kod, samt belysa eventuella begränsningar och nyttor med att standardisera och automatisera kodkommentarer. Baserat på avhandlingens syfte har följande forskningsfråga utvecklats:

- Hur kan användningen av Large Language Models underlätta för programmerare att kommentera källkod?

1.6 Förväntat resultat

Resultaten från den här studien förväntas bidra till att belysa fördelarna och utmaningarna med att använda LLMs för automatisering av kodkommentarer och även att identifiera eventuella förbättringsområden för befintlig modell. Studien kan också ge insikt i hur utvecklare reagerar på och använder automatiserade kodkommentarer och hur dessa kommentarer påverkar koden och dess förvaltning. Genom att utforska möjligheterna med LLMs för automatisering av kodkommentarer kan denna avhandling bidra till att förbättra effektiviteten och kvaliteten i mjukvaruutveckling samt minska arbetsbördan för utvecklare.

1.7 Målgrupp

Denna studie har en bred tillämpning inom hela området för mjukvaruutveckling och är speciellt inriktad på intressenter av OpenAIs ChatGPT, särskilt programmerare. Företag inom utvecklingssektorn utgör en viktig målgrupp, då de har potential att dra nytta av att implementera ChatGPT i sina verksamheter. Dessutom vänder sig studien till forskare och utvecklare inom området för Large Language Models, som strävar efter att vidareutveckla ChatGPT och utforska dess möjligheter.

1.8 Avgränsning

Den här studien undersöker huruvida nuvarande Large Language Models kan hjälpa programmerare att kommentera källkod. Skribenterna har valt att fokusera på innebörden för rollen som programmerare på individnivå, vid tillämpning av tjänsten OpenAI ChatGPT. Eftersom studiens fokus berör rollen på individnivå, kommer det kontextuella sammanhanget att begränsas till detta och därmed inte inkludera ett verksamhetskontext. Vidare avgränsning för studien är att fokusera på en specifik aspekt inom programmering, vilket är kommentering av kod.

1.9 Disposition

Rapporten är strukturerad i sex kapitel som omfattar en introduktion, en beskrivning av tillvägagångssättet, ett teoretiskt ramverk, insamlat empiriskt material, en analytisk del samt en avslutande diskussion och slutsatser.

I introduktionskapitlet presenteras ämnet för studien, följt av en introduktion till problemområdet och syftet samt forskningsfråga. Kapitlet klargör också studiens omfattning och avgränsningar.

Nästa kapitel kallas metod och genomförande. Som namnet antyder, beskriver kapitlet hur studien har genomförts och vilka metoder som har använts. Arbetet innefattar observationer, intervjuer och litteraturstudier. Kapitlet beskriver även validiteten och reliabiliteten i studien.

Vidare presenteras den valda teorin för arbetet i det teoretiska ramverket. Teori inom ämnen som Artificiell intelligens, Large language models, ChatGPT, kodkonventioner, utmaningar och begränsningar med ChatGPT presenteras för att kunna besvara forskningsfrågorna.

I det följande kapitlet, empiri, presenteras de respondenter som har valts ut som grund för studien, tillsammans med beskrivning av tillvägagångssätt och resultaten från utförda intervjuer och observationer.

I analyskapitlet utvärderas den bearbetade datan och jämförs med teorin för att komma fram till studiens resultat. Analysen lyfts sedan till en generell nivå och besvarar därmed forskningsfrågorna.

Avslutningsvis presenteras diskussionen och slutsatsen av hela studiens resultat i det sjätte kapitlet. En utvärdering av arbetet presenteras tillsammans med rekommendationer och förslag på vidare forskning inom ämnet.

2. Metod

Detta kapitel introducerar och motiverar valet av forskningsmetoder för studien. Avsnittet omfattar information om källor för datainsamling och avslutas med en diskussion om tillförlitligheten av det insamlade materialet.

2.1 Forskningsstrategi

Följande metoder har använts för att besvara studiens frågeställning:
Litteraturstudier, observationer och intervjuer *(Vidare beskrivning av respektive i avsnitt 2.4)*.
Genom litteraturstudier har relevant information samlats in och placerats som grund till studiens frågeställning och det teoretiska ramverket. Intervjuer och observationer har använts på ett explorativt sätt för att fylla på det gap som finns inom teorin, samt att jämföra utfall mot relevant sekundärdata. Studien omfattades av en inledande intervju med respektive respondent för att fastställa grundläggande information och därmed skapa underlag inför kommande observationer. En avslutande intervju genomfördes därefter för att erhålla underlag till att besvara frågeställningarna.

För att besvara uppsatsens frågeställning krävs både verbal och visuell data i form av intervjuer och observationer. Uppsatsen kräver därmed främst en kvalitativ ansats. Utvecklingen av det teoretiska ramverket har skett abduktivt och iterativt. Den kvalitativa datainsamlingen har därmed ökat skribenternas förståelse kring ämnet, samt gett upphov till ändrad riktning för insamling av litteratur. Genom att låta arbetet vara abduktivt tillåts skribenterna anpassa teorin till utvecklingen av datainsamlingen. Således har risken minskat avsevärt att studien låter teori vara den avgörande faktorn för att beskriva verkligheten (Patel & Davidson, 2011).

Syftet med studien var att undersöka huruvida Large Language Models kan hjälpa mjukvaruutvecklare att kommentera källkod. För att hjälpa till att besvara frågeställningen har litteraturstudier genomförts kring Artificiell intelligens (AI), Large Language Models (LLM), kodkonventioner och utmaningar, samt begränsningar med ChatGPT.

Sammanfattningsvis är en kvalitativ ansats att föredra för att besvara uppsatsens frågeställning, då den möjliggör en djupgående undersökning av både verbal och visuell data genom intervjuer och observationer. Dessutom har den kvalitativa datainsamlingen spelat en central roll för att öka förståelsen av ämnet och har bidragit till abduktiv och iterativ utveckling av det teoretiska ramverket.

2.2 Urval av respondenter

Urvalet av respondenter har genomförts med hänsyn till deras erfarenheter inom ämnesområdet. Väsentliga variabler såsom ålder, utbildning och tjänsteställning har använts som viktiga parametrar för att forma urvalet. Denna noggranna selektering var nödvändig eftersom den kvalitativa undersökningen krävde att respondenterna besitter specifik kompetens och kunskap inom ämnet. Samtidigt är respondenterna inom urvalet liknande när det gäller deras bakgrund inom IT, men arbetar inom olika sektioner eller områden inom mjukvaruutveckling.

Ålder har en viktig betydelse då den påverkar respondenternas samlade erfarenhetsnivå och ger också en bredare inblick i hur tekniken har utvecklats över tid. Tjänsteställningar avser respondenternas nuvarande roller inom mjukvaruutveckling och ger insikt i deras arbetsansvar och eventuell interaktion med olika människor inom större team. Detta ger grund för att diskutera och förstå hur mjukvaruutveckling praktiseras och genomförs. Vidare har urvalet även inkluderat en person med mindre erfarenhet inom arbetslivet för att tillföra ytterligare nyanserade perspektiv när det gäller ChatGPT och kommentering av kod.

För genomförandet och insamlingen av empirisk data har olika tekniker tillämpats, främst intervjuer och observationer. Dessa metodologier har möjliggjort en djupare förståelse av respondenternas tankar, åsikter och beteenden i samband med ämnet som studeras. Genom att kombinera dessa metoder kan forskaren få en helhetsbild av respondenternas erfarenheter och synpunkter, vilket har bidragit till en mer nyanserad och omfattande analys av studiens frågeställningar.

Bild 1

Respondent	Ålder	Område	Tjänsteställning	Programmeringsspråk
A	32	Webbutveckling	Senior	PHP
B	33	Mobila nätverk	Senior	Erlang
C	23	Webbutveckling	Junior	Javascript
D	32	Embedded systems	Senior	Python

2.3 Datainsamling

Grund för studien omfattas av litteraturstudier samt insamling av empirisk data i form av observationer och intervjuer.

2.3.1 Litteraturstudier

Syftet med litteraturstudien är att hitta relevant information som kan härledas till frågeställningen. Detta i avseende att identifiera olika och/eller liknande infallsvinklar. För att identifiera relevanta litteraturstudier har följande sökkombinationer främst inkluderat följande termer: AI, LLM, ChatGPT, OpenAI, automatisering, riktlinjer vid kommentering av kod etc. Strategin för att välja ut artiklar har innefattat sökning av litteratur via olika databaser som Proquest, Diva, Google Scholar och Primo. Genom ovan nämnda sökkombinationer har relevant litteratur identifierats. Litteratur har främst valts ut baserat på antal relevanta citeringar som kan härledas till studiens syfte, samt att de är peer reviewed. Andra tillvägagångssätt för att välja ut, samt bedöma relevansen av litteratur har inneburit att läsa i litteraturens introduktion samt att se över innehållsförteckningen i relation till studiens primära frågeställningar. Vidare har sökt litteratur gett upphov till snöbollseffekten, eftersom att referenshänvisningar till andra vetenskapliga teorier och metoder har hittats och därefter granskats (Noy, 2008).

2.3.2 Intervjuer

Kvalitativa intervjuer handlar om att upptäcka och identifiera egenskaper och karaktäristika hos något, exempelvis livsvärlden eller åsikter om ett fenomen (Patel & Davidson, 2011). Studiens syfte i förhållande till intervjuer, var att identifiera innebörden samt behov av att kommentera kod. Därefter undersöktes hur Large Language Models kan användas som hjälpmedel för att kommentera källkod. Detta genomfördes genom semistrukturerade intervjuer med studiens utvalda respondenter. Vid upprepade intervjuer är det möjligt för respondenter att förändra sin uppfattning, få nya insikter eller att de lär sig något nytt. För att kunna förstå de faktorer som påverkar respondentens ändrade åsikt är det viktigt att ta hänsyn till variabler, som exempelvis tidigare genomförda intervjuer (Patel & Davidson, 2011). Under datainsamlingen och arbetets gång kommer eventuellt differentierade tankar att dyka upp som berör hela problemområdet.

För att erhålla ett levande förhållande till datainsamling så har löpande analyser av insamlad data genomförts (Patel & Davidson, 2011). Vidare belyser Patel & Davidson (2011) att fördelen med löpande analys är att det kan ge upphov till nya idéer och insikt inför fortsatt arbete. Eventuellt får skribenterna insikt att vissa saker saknas i intervjun, eller att respondenten uppfattar frågor på ett sätt som författarna inte hade tänkt sig. Löpande analys är en faktor som skiljer kvantitativ- och kvalitativ bearbetning åt. I kvantitativa studier bearbetas vanligtvis datan när allt material är insamlat (Patel & Davidson, 2011).

2.3.3 Observationer

Studien kräver en hög grad av empiriskt underlag genom observationer. Observationerna ska vara planerade systematiskt för att samla in information inom området. Genom observation kan beteenden och skeenden samlas in i relativt naturligt sammanhang i samma stund som de

inträffar (Patel & Davidson, 2011). Respondenterna kommer följa ett i förväg bestämt observationsschema, vilket innebär att det handlar om strukturerade observationer. Enligt Patel & Davidson (2011) så bör observatörer ta ställning till följande frågor: Vad ska observeras? Hur ska vi registrera observationerna? Hur ska vi som observatörer förhålla oss? Detta beaktades vid utförandet.

Under observation har samtliga respondenter blivit ombedda att använda en förbestämd, omarbetad och specifik instruktion att skriva in i ChatGPT. Instruktionen har använts för att tydligt beskriva hur ChatGPT skulle förhålla sig till nästkommande inmatning. Inmatningen bestod av ett kodstycke på omkring 50-100 rader okommenterad kod, som respondenterna valde ut enligt specifika kriterier. Grund till kriterierna var det programmeringsspråk respondenterna använder främst, samt att kodstycket skulle vara relativt komplext.

Respondenterna hade fullständig befogenhet att bedöma komplexiteten hos det valda kodstycket. Utförande och resultat observerades genom att respektive respondent skärmdelade sina utföranden via Zoom. Därefter ombads respondenterna att analysera ChatGPTs output. Detta eftersom respondenterna har spetskompetens att utvärdera och avgöra huruvida ChatGPTs framställning av kommentarer stämde överens med kodstycket. Vidare fick respondenterna reflektera över eventuella brister, men även påpeka positiva fördelar med att använda ChatGPT som hjälpmedel vid kommentering av kod. För att uppnå hög tillförlitlighet analyserades varje respondentintervju tillsammans med de genomförda observationerna.

2.4 Analys av empirisk data

Vid analys av de svar som samlats in i samband med de semistrukturerade intervjuerna som ägde rum, har en kvalitativ innehållsanalys applicerats. Syftet med detta är att kunna se samband med teorier kopplade till studiens forskning. Vidare är målet att se huruvida den nyproducerade datan stärker och svarar på forskningsfrågan.
Empirisk data samlades in genom intervjuer och observationer, vilka därefter analyserades för att ge en mer fullständig bild. Resultaten från dessa olika tillvägagångssätt kan sammanfalla eller peka åt olika håll. En triangulering av insamlad data möjliggörs därmed, vad gäller litteraturstudier, intervjuer och observationer (Patel & Davidson, 2011). Litteratursökningen kommer att utgöra en bra grund för studien, medan insamling av data genom intervjuer och observationer kommer att användas som styrparameter för att bedöma relevans och eventuell omarbetning av det teoretiska ramverket.

2.5 Metodreflektion

Valet av metod baserades på vad studien ämnade att utforska. I förhållande till frågeställningen ansågs en kvalitativ undersökning relevant för att kunna bedriva en pålitlig och verklighetsbaserad datainsamling på individnivå. Den litteratur som använts i studien hjälper till att skildra AI historiskt, i nutid, men även ge en uppfattning om vart den är på väg.

Med fokus på LLMs och i synnerhet kommentering av kod, har skribenterna via intervjuer och observationer kunnat stärka de argument som litteraturen tagit upp, genom att bekräfta att specifika scenarion utspelar sig på ett visst sätt i verkligheten. Utifrån detta kunde skribenterna säkerställa att studiens resultat inte blivit styrda av teorier och antaganden, utan faktiska händelser som blivit väldokumenterade.

Genom att bedriva en kvalitativ studie, tillkommer för- och nackdelar. Vid framtagningen av intervjufrågorna ämnade skribenterna att ge upphov till intervjuobjekten att kunna tala fritt. Frågorna var tänkta att vara utformade på ett sådant sätt att de inte är ledande på något vis. Detta för att garantera att svaren är säregna och bidrar med unika infallsvinklar som litteraturen inte har möjlighet till.

På grund av valet av intervjuobjekt kunde trovärdigheten till studien öka avsevärt, detta då personerna i fråga besitter mångårig erfarenhet inom ett relevant avsett område och har en auktoritär position. Vidare är det förstahandsinformation som behandlats i både verbal och visuell form, där man kunnat observera hur intervjuobjekten agerade i förhållande till användning av ChatGPT. Då intervjuerna var tänkta att påvisa hur respondenterna hade använt AI-modellen yrkesmässigt, kunde validiteten öka till insamlad data. Detta eftersom det potentiellt kan återspegla hur en vanlig arbetsdag kan se ut, med ChatGPT som hjälpmedel vid kommentering av kod.

Beroende på följderna av anvisningarna som tillgavs respondenterna kunde följdfrågor och oväntade händelser uppkomma. Detta gjorde undersökningen mer flexibel, men utgjorde även en utmaning. All insamlad data är inte värdefull och en av utmaningarna var att försöka få ut värde utifrån samtliga frågor och observationer, samt att vara förenlig med studiens ämne och frågeställning.

En potentiellt negativ aspekt med handplockade respondenter är att resultatet riskerar att bli vinklat. Detta då intervjuobjekten i fråga har liknande arbetsuppgifter och på grund av deras miljö riskerar dem automatiskt att dela åsikter som andra deltagare. En utmaning blir därmed att få en egenartad åsikt som kan överföras till studiens resultat, utan att äventyra resultatets tilltro. För att motverka detta intervjuades och observerades respondenterna enskilt, och därmed erhölls egenartade åsikter som kunde tillskrivas studiens resultat, med hög tilltro.

Semistrukturerade intervjuer kan innebära en utmaning då respondenterna tenderar att vara mindre objektiva, vilket kan leda till mer personliga åsikter och subjektiva svar. Detta kan påverka svaren negativt genom att minska både trovärdighet och nyansering. För att undvika oklarheter är det avgörande att intervjuaren har rätt fokus och styr intervjun i rätt riktning.

2.5.1 Reliabilitet

Jacobsen (2017) pekar på begreppet "Reliabilitet" som är synonymt med ordet "trovärdighet". Denna punkt ämnar att belysa studiens tillförlitlighet och att man via lämpliga, korrekta metoder säkerställer information som reliabel. Intervjuaren ämnar att erhålla ett objektivt förhållningssätt för att undvika att påverka det som undersöks i studien.

För att säkerställa detta kommer intervjuerna att spelas in för att ge upphov till transkribering. Vidare bifogas en intervjuguide för att ytterligare stärka förtroendet och ge läsaren en insyn i hur man gått tillväga vid datainsamlingen.

Reliabilitet bör enligt Patel & Davidson (2011) ses mot bakgrund av den unika situation som råder vid undersökningstillfället. Enligt Skärvad och Lundahl (2016) uppnås god reliabilitet då det råder ett oberoende mellan observatören och den rådande situationen. Därav placeras stor vikt vid att vara neutral och inte ta något aktivt ställningstagande.

2.5.2 Validitet

Validiteten förstärks genom att skribenterna reflekterar och är medvetna om de val som görs i hantering av information, samt hur detta kan påverka analysen (Patel & Davidson, 2011).
En studies validitet kan delas upp i två spektrum, varav den ena är intern och den andra är extern. Jacobsen (2017) menar att en av faktorerna kring studiens trovärdighet handlar om huruvida insamlade resultat uppfattas som trovärdiga eller inte. I syfte att säkerställa studiens interna validitet ämnar författarna att göra en noggrann bedömning av lämpliga intervjuobjekt. Faktorer som man tar i beaktning vid urvalet är:

- Primärkälla
- Erfarenhet
- Relevans till studiens frågeställningar och syfte

Vidare menar Jacobsen (2017) att den externa validiteten kan kopplas till andra människor utöver de som blivit handplockade till studien. Med andra ord, ifall studiens resultat kan generaliseras till den breda massan.

2.5.3 Generaliserbarhet

Enligt Patel & Davidson (2011) avser generaliserbarhet förmågan att dra slutsatser och tillämpa resultat och kunskap från en specifik studie eller situation på en bredare population eller inom liknande kontext (Patel & Davidson, 2011). Denna studie har en begränsad generaliserbarhet på grund av dess specifika fokus på en enskild tjänst och relativt innovativt

tillvägagångssätt. Urvalet av respondenter baserades huvudsakligen på deras expertis inom olika programmeringsspråk. Det är därför inte möjligt att generalisera studiens tillvägagångssätt och dra slutsatser som är tillämpliga för alla befintliga programmeringsspråk och tillhörande grupper av programmerare. Resultat av studien kan dock ha betydelse för hur Large Language Models kan tillämpas inom samtliga programmeringsspråk som omfattas av studien.

2.5.4 Konsekvenser av avgränsning

Genom att begränsa fokus till en enskild aspekt av programmering, till exempel kommentering av kod, kan det uppstå en förenklad uppfattning om potentiella konsekvenser. Programmeringsarbete omfattar en bred variation av uppgifter och utmaningar som kan påverkas på olika sätt av LLMs. Genom att enbart rikta in sig på kommentering av kod kan det finnas andra aspekter av programmeringsuppgifter som inte beaktas och därmed inte kan återspeglas i resultaten.

Genom att fokusera endast på individnivån och begränsa det kontextuella sammanhanget kan betydelsefulla faktorer och interaktioner som påverkar programmerarnas upplevelser och konsekvenser på gruppnivå eller organisatorisk nivå negligeras. Detta kan medföra en förlust av kontextuell relevans för studiens resultat. Det finns också en risk att perspektiv och erfarenheter från andra intressenter, såsom arbetsgivare, projektledare eller användare av den kod som programmerarna skapar, inte fångas upp. Det kan vara värdefullt att inkludera en bredare variation av perspektiv för att erhålla en mer holistisk förståelse av konsekvenserna av LLM.

3. Teoretiskt ramverk

3.1 ChatGPT, dess bakgrund och funktion

De senaste åren har framstegen inom naturlig språkbehandling som uppnåtts av stora språkmodeller (LLM-er) varit det mest anmärkningsvärda genombrottet inom AI-forskning. OpenAI grundades år 2015 av bland andra Elon Musk och Sam Altman. Det är en organisation som ligger till grund för utvecklingen av ChatGPT. Med dess fokus på att utveckla artificial general intelligence (AGI) har man som mål att skapa olika typer av AI, för att hjälpa människor och underlätta arbetsuppgifter såväl vardagliga behov (Ray, 2023). ChatGPT har fram till att den officiellt lanserades till allmänheten i beta stadiet den 30 november 2022 genomgått flera olika versioner och uppdateringar.

I publikationen *Internet of Things and Cyber-Physical Systems Volume 3, 2023, (s. 121-154)* beskrivs de olika versionerna och hur de kontinuerligt förbättrats genom åren. Nedan är en sammanställning av AI-modellens historiska skildring.

1. GPT-1 (Generative Pre-trained Transformer): Detta var den ursprungliga versionen av GPT som släpptes av OpenAI år 2018. Den använde transformerarkitekturen och tränades på stora mängder textdata för att generera koherent text baserat på givna inmatningar. Transformerarkitekturen, introducerades i artikeln "*Attention is all you need*" av Vaswani et al (2017) och utgör en form av neural nätverksarkitektur som använder uppmärksamhetsmekanismer för att effektivt hantera sekvensdata. Transformerarkitekturen har haft en betydande påverkan inom området för naturlig språkbehandling och används inom en rad olika tillämpningar (Bubeck et al, 2023).
2. GPT-2: Lanserad 2019, GPT-2 var en förbättrad version av GPT-1 och blev snabbt känd för sin förmåga att generera kreativ och realistisk text. GPT-2 var betydligt större än sin föregångare och tränades på ett massivt dataset med olika källor från internet.
3. GPT-3: Introducerad under 2020, GPT-3 var en revolutionerande uppdatering som imponerade med sin enorma storlek och imponerande förmågor. Den hade 175 miljarder parametrar och kunde utföra olika språkliga uppgifter såsom översättning, textgenerering och svar på frågor. GPT-3 fick mycket uppmärksamhet för sina förmågor, men det fanns också diskussioner om dess begränsningar och behovet av kontrollmekanismer.
4. GPT-3.5: En fortsättning på GPT-3 som presenterades under 2021, GPT-3.5 inkluderade förbättringar och justeringar för att förbättra modellens prestanda och hantera några av de tidigare identifierade utmaningarna. Det tillhandahöll en högre grad av kontroll över textgenereringen och implementerade fler språkstilar och anpassningsalternativ.
5. GPT-4: Den nyaste upplagan av GPT som uppvisat förbättrat fakta, styrbarhet och följsamhet. OpenAI har gjort betydande framsteg inom djupinlärning genom att

> introducera GPT-4. Till skillnad från sina föregångare, kan GPT-4 bearbeta både text- och bild-inmatningar, samt generera text- och bild-utdata. Även om det kanske inte matchar mänskliga förmågor i verkliga scenarier, har GPT-4 uppnått prestanda på mänsklig nivå på olika professionella och akademiska riktmärken.

GPT-modeller är utformade för att generera sammanhängande text i naturligt språk, såsom meningar, stycken och dokument, som är i linje med mänskligt språk. De tränas på stora mängder textdata utan övervakning och lär sig att förutsäga nästa ord i en sekvens av text. Genom finslipning på specifika uppgifter kan modellerna anpassas för textklassificering eller frågor och svar. Denna kombination av förträning och finslipning möjliggör att GPT-modeller kan generera kvalitativ text baserat på givna inmatningar (Ray, 2023).

En undersökning genomförd av Microsoft testar OpenAIs senaste version, ChatGPT-4 enligt olika kriterier och jämför kapaciteten hos tidigare versioner av ChatGPT. ChatGPT-4 har förmågan att lösa nya och svåra uppgifter inom matematik, programmering, bildbehandling, medicin, juridik, psykologi och mer, utan att behöva någon särskild instruktion. Dess prestation i alla dessa uppgifter är anmärkningsvärt nära mänsklig prestation och överträffar ofta tidigare modeller av ChatGPT. Forskare inom AI har arbetat med stora språkmodeller (LLM) som uppvisar imponerande förmågor inom olika områden och uppgifter, vilket sträcker sig utanför gränserna för vår kunskap om inlärning och kognition. OpenAIs senaste modell, GPT-4, tränades med en aldrig tidigare skådad mängd beräkningskraft och data (Bubeck et al, 2023). Med tanke på bredden och djupet av GPT-4:s förmågor tror Bubeck et al (2023) att det rimligen kan betraktas som en tidig (men fortfarande ofullständig) version av ett artificiellt allmänt intelligenssystem (AGI).

3.2 Kommentering av kod

Mjukvaruutvecklare arbetar till stor del med att underhålla, förstå och bli familjär med befintlig källkod. Underhållsfasen, som innehåller bland annat felsökning och uppdateringar, är vanligtvis den mest kostsamma fasen inom mjukvaruutveckling. Genom att öka förståelsen av programkoden kan man minska den nödvändiga tiden för underhåll och sänka risken för att introducera nya buggar (Nielebock et al, 2019). Programförståelse är ett forskningsområde som undersöker hur utvecklare förstår befintliga program. Flera mönster har utvecklats för att förbättra källkod i program och därmed dess förståelse, till exempel riktlinjer för ren kod eller designmönster. Dock uppstår dessa enligt Nielebock et al (2019) & Huang et al (2020) ofta från personliga preferenser och erfarenheter snarare än vetenskapliga metoder.

Att kommentera kod är en de facto standard inom de flesta programmeringsspråk och används för att låta utvecklare beskriva kod med egna ord. Följaktligen ger de en extra mekanism för att förbättra förståelsen av koden (Nielebock et al, 2019 & Skansholm, 2014 & Huang et al, 2020). Källkod exekveras av datorer, men det är viktigt för utvecklare att förstå innebörden. Väl dokumenterad kod kan öka förståelsen av implementerad kod på grund av den extra information som delges genom kommentarer. Trots det är det sällan som uppdaterad kod innefattar uppdatering av befintliga kommentarer. Detta kan enligt Nielebock

et al (2019) bero på tidsbrist, låg motivation eller omstrukturering av kod. Inom mjukvaruutveckling syftar omstrukturering av kod eller refaktorisering till att gradvis förbättra kvaliteten på programkoden. Detta görs enligt Skansholm (2014) för att systematisera koden och göra den enklare att förstå och hantera. Genom att rensa upp i koden kan den göras mer lättförståelig, underhållsvänlig och utvecklingsbar. Tidigare ej uppdaterade kommentarer kan då eventuellt motverka förståelsen av befintlig kod (Skansholm, 2014).

Automatiska verktyg som påminner mjukvaruutvecklare att lägga till kommentarer till skriven kod innebär inte att programmeraren helt överlåter ansvaret till verktyget. Mjukvaruutvecklare behöver skapa passande kommentarer och välja lämplig plats för kommentarer i källkoden. Enligt Huang et al (2020) är det viktigt att inte överanvända kommentarer i koden eftersom för många kommentarer kan störa utvecklarna och kräva extra arbete. Kommentarer bör därför användas sparsamt och endast när det är nödvändigt för att förklara komplex kod eller logik.

3.2.1 Kodkonventioner

Enligt Nielebock et al (2019) finns det typiskt tre sätt att kommentera kod: Icke kommenterad kod, kod med implementeringskommentarer och kod med dokumentationskommentarer.
Icke kommenterad kod avser helt enkelt källkod som helt saknar kommentarer.
Implementeringskommentarer avser kommentarer som beskriver en eller flera rader kod. Det kan exempelvis innefatta vad en specifik funktion är till för, beskrivning av algoritmer, eller kommentarer som beskriver förekomst av buggar. Dessa kommentarer förekommer enligt Nielebock et al (2019) oftast som enskilda rader, precis före den kod som kommentaren avser.
Dokumentationskommentarer beskriver främst konstruktioner, som exempelvis interfaces, metoder eller klasser/typer. Denna form av kommentarer beskriver konstruktioner på ett mer översiktligt sätt, för att ge nödvändig förståelse för utförande av konstruktionerna, och beskriver inte några konkreta implementeringsdetaljer.
Implementeringskommentarer avser rad 8,11 och 13, dokumentationskommentarer avser rad 1 - 6 (se bild 2 nedan).

Bild 2

```
1  /**
2  * Calculate the sum of all roman numerals
3  * in an array as an integer
4  * @param strings - array with roman numerals
5  * @return - sum of roman numerals as integer
6  */
7  public int foo(String[] strings) {
8    // variable to accumulate converted roman numerals
9    int number = 0;
10
11   // iterate over all roman numerals
12   for(int i = 0; i < strings.length; i++){
13       // add current roman numeral to result
14       number += convertRoman2Int(strings[i]);
15     }
16   return number;
17 }
```

Empirical Software Engineering (2019) 24:1418–1457 (s. 1421)

3.2.2 Python

PEP 8 - Style Guide for Python code, skriven av Rossum et al (2001) är ett dokument som ger kodkonventioner för Python som utgör standardbiblioteket för Python. Avsnittet i dokumentet som avser kommentering av kod beskriver riktlinjer som bör efterföljas när man kommenterar källkod. Det är enligt Rossum et al (2001) sämre att använda kommentarer som motsäger källkoden. Det är därför viktigt att uppdatera kommentarer i med att källkoden uppdateras. Detta är något som Nielebock et al (2019) också lyfter som en kritisk faktor. Vidare generella riktlinjer är att kommentarer bör vara fullständiga meningar, där första bokstaven ska vara i versal, förutom vid de fall då det är en identifierare som börjar med gemener. Följaktligen ska man aldrig ändra på identifierare (Rossum et al, 2001). PEP 8 tar upp två typer av kommentarer, block kommentarer samt inline kommentarer. Dessa liknar sig vid dokumentationskommentarer respektive implementeringskommentarer som Nielebock et al (2019) beskriver. Det som skiljer sig är att PEP 8 implementeringskommentarer (inline kommentarer) skrivs på samma rad som den källkod som ska beskrivas, och inte över. Enligt Rossum et al (2001) ska implementeringskommentarer användas väldigt sparsamt, eftersom de kan utgöra en distraktion vid de fall som källkoden är självbeskrivande. Likaså menar Huang et al (2020) att dessa endast bör användas för att förklara komplex kod eller logik.

3.2.3 PHP

I PHP används både inline- och multiline kommentarer. Manualen för PHP (php.net/manual) beskriver vissa generella riktlinjer för kommentering av kod. Man bör kommentera alla funktioner, klasser och metoder samt kommentarer på globala variabler som används. Detta gäller främst för kod som inte är helt självklar, likt vad Huang et al (2020) beskriver som “komplex kod”. Det finns syntax för att kommentera en rad, samt för att kommentera flera rader.

3.2.4 JavaScript

Det finns ingen direkt officiell standard i Javascript för kommentering av kod. Det förekommer många olika Javascript-bibliotek och CMS (Content Management System) som har olika standards och riktlinjer. Ett av de kanske mest kända CMS är Wordpress. Enligt Wordpress.org (2023) ska kommentarerna komma före koden som de hänvisar till och bör föregås av en tom rad. Stora bokstäver ska användas för den första bokstaven i kommentaren och en punkt bör inkluderas i slutet, vid de fall som det är en fullständig mening. Det ska också finnas ett mellanslag mellan kommentartecknet och kommentaren.Som undantag tillåts inline-kommentarer när de används för att ange särskilda argument i formella parameterlistor.

3.2.5 ERLANG

För att kommentera kod i ERLANG börjar man med procenttecken (%), därefter följs kommentarer till slutet på specifik rad. Det finns inga block kommentarer i ERLANG. Det finns inte heller någon syntax för att skapa multiline kommentarer, utan man får kommentera bort varje rad genom att ange ´%´ (Erlang.org, 2023).

3.3 Konsekvenser vid bristande kommentering av kod

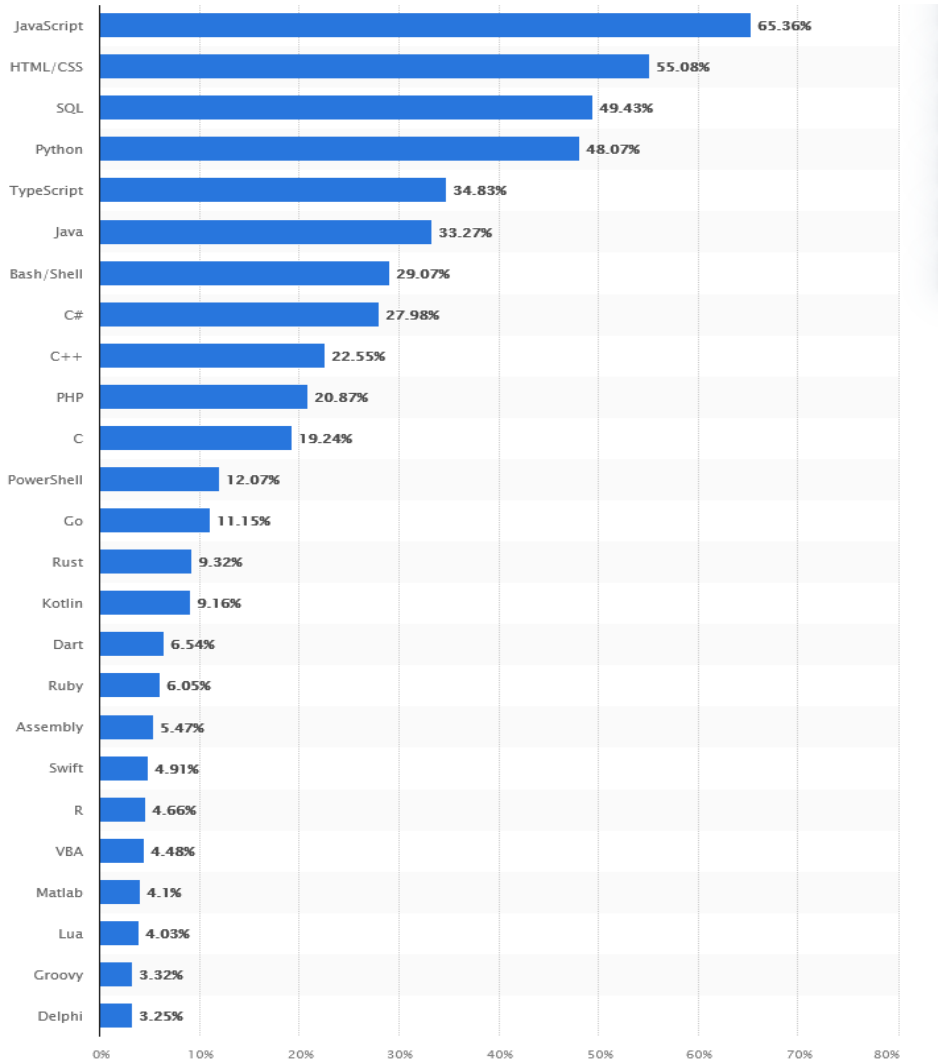
Stora projekt kräver ofta samarbete mellan flera utvecklare under längre perioder. Ett av de större problemen enligt Laramee (2013) är när källkod skrivs utan att man följer konventioner. Detta medför att det är svårare att återanvända kod, vilket gör att utvecklare återuppfinner hjulet gång på gång. Att skriva svårläst kod är enkelt och tyvärr ofta standard för olika utvecklare, dessutom har professionella programmerare enligt Salviulo & Scanniello (2014) tendenser att ignorera kommentarer i källkod. Detta kan leda till att koden blir svårtolkad, även för den utvecklaren som initialt skrev källkoden. Andra komplikationer uppstår då utvecklare ska uppdatera befintlig kod och har svårigheter att sätta sig in i strukturen eftersom kommentarer saknas och/eller är föråldrade (Laramee, 2013). Samtidigt så kan förhållanden i projekt vara tidsbrist och låg motivation, som Nielebock et al (2019) belyser. Enligt Laramee (2013) så utgör kodkonventioner, som även innefattar hur man kommenterar kod, en bra grund för att lyckas med mjukvaruutveckling. Detta eftersom individer i olika teams, och även personer utanför, kan förstå och förvalta koden. Kommentering av kod har framförallt större betydelse för nybörjare som kan använda den extra informationen för att öka förståelsen för olika program (Salviulo & Scanniello, 2014). Enligt Smit et al (2011) är en tydligt definierad namngivningsstil som matchar befintliga bibliotek en stor hjälp till att koppla samman syntax med semantisk betydelse. En implicit följd är att bra kod gör det lättare att förstå designen. Välformulerade kommentarer är till stor hjälp för nya utvecklare att förstå sig på befintlig källkod, samt att öka samstämmigheten inom organisationer, teams och att underlätta användning av olika API (Smit et al, 2011).

Varje programmeringsspråk har en viss syntax vad gäller kommentering av kod. Dock verkar generella riktlinjer vara relativt likartade, oavsett vilket språk det handlar om. Det handlar mer om hur man väljer att kommentera inom en viss organisation, projekt eller team, snarare än det programmeringsspråk man använder. Undantaget är ERLANG, där det inte finns en specifik syntax för att göra fler rad kommentarer. Sammanfattningsvis finns det generellt sett fyra olika sätt att kommentera kod i majoriteten av programmeringsspråk. Single-line kommentering, multi-line kommentering, dokumentationskommentering och inline kommentering.

3.4 Popularitet bland programmeringsspråk

En undersökning publicerades av företaget Statista i juli månad 2022, med syfte att kartlägga de mest populära programmeringsspråken runt om i världen. Baserat på svaren av 71,547 respondenter kunde man fastställa att JavaScript i särklass var det mest populära programmeringsspråket, med totalt 65.36% användare. Detta följdes upp av HTML/CSS som hade 55.08% och sedan SQL med 49.43% Vailshery (2022). Se **bild 3** för mer detaljerad vy. Med hjälp av nedanstående graf som visar populariteten av olika programmeringsspråk, kommer studien även att undersöka om det finns en korrelation mellan popularitet, kontra hur väl ChatGPT kommenterar kod. Som tidigare nämnt, tränas LLMs med hjälp av användarinput och kan med tiden få bättre prestanda beroende på mängden data den tillhandahåller. OpenAIs senaste modell, GPT-4, tränades med en aldrig tidigare skådad mängd beräkningskraft och data (Bubeck et al, 2023).

Bild 3



(Statista, 2020)

3.5 Utmaningar med ChatGPT

AI har på senare år utvecklats till att vara en grundpelare inom vårt samhälle idag. Med dess enorma bredd och olika användningsområden är det ett verktyg som gör våra vardagliga liv bekvämare, men som även är en nödvändighet för att vara konkurrenskraftig inom diverse arbetsbranscher (Ray, 2023). ChatGPT har sedan den lanserades genomgått olika faser och uppdateringar. Trots det kvarstår vissa komplikationer vid användning av tjänsten. I publikationen *"Internet of Things and Cyber-Physical System" (s. 121-154)* adresserar man ett antal utmaningar som ChatGPT har kopplat till användning inom arbets- och forskningsfältet.

En av dessa är att den än idag har problem med pålitlighet och noggrannhet. Enligt Ray (2023) & Bubeck et al (2023) producerar ChatGPT emellanåt inkorrekt eller missledande information och data. Detta innebär att tillförlitligheten för applikationen minskar, samt att användandet inom arbets- och forskningsrelaterade områden begränsas avsevärt. För att man ska kunna använda AI-genererat innehåll bör man se över denna punkt, i syfte att bibehålla integritet som är en kritisk faktor inom vetenskaplig forskning (Ray, 2023). ChatGPT tillhandahåller en enorm mängd data som den sedan omvandlar till svar för användaren. Källmaterialet kan innehålla partiska åsikter eller fördomar, vilket i sin tur kan spridas vidare oavsiktligt av applikationen. Konsekvenserna blir att vetenskaplig forskning som använder sig av ChatGPT som hjälpmedel, riskerar att påverka dess riktning (Ray, 2023). Ytterligare problematik som AI-modellen potentiellt kan medföra är att det finns en risk att människor förlitar sig för mycket på ChatGPT. Följderna blir att kritiskt tänkande och självständiga problemlösningar minskar bland personer som använder det inom sina yrken, men även bland forskare (Ray, 2023).

3.6 ChatGPT som hjälpmedel inom programmering

LLMs är AI-modeller som via stora mängder textdata lär sig att förstå och skapa mänskligt språk. De är en sammanställning av neurala nätverk och maskininlärningsalgoritmer, i syfte att bearbeta språk på ett snarlikt sätt som människor gör (Ray, 2023). LLMs används idag inom ett brett spektrum av tillämpningar. Det inkluderar bland annat språköversättningar, chatbots och textsammanfattning. Några arbetsområden som dessa AI-modeller verkar i är inom ekonomi, sjukvård och utbildning (Ray, 2023). Nedan listas tre sätt som ChatGPT kan användas som hjälpmedel inom programmering.

1. ChatGPT har visat sig vara användbart i syfte att generera kodavsnitt baserat på användarinmatning. AI-modellen har förmågan att förse användaren med kod som kan användas till ett särskilt ändamål, som exempelvis att implementera en specifik funktion. ChatGPT gör detta genom att analysera data från det specifika programmeringsspråket som är tänkt att användas, samt dess funktioner och krav (Kashefi & Mukerji, 2023).

2. ChatGPT kan användas som hjälpmedel inom dokumentation av kod. AI-modellen analyserar data om programmeringsspråket, kodstruktur och funktionskrav. Det ger upphov till applikationen att rekommendera bästa praxis och standarder inom koddokumentation. ChatGPT underlättar processen att kommentera kod, genom att forma en tydlig och koncis dokumentation som är enkel att förstå för läsaren (Haleem et al., 2023).

3. ChatGPT kan hjälpa användaren att på ett mer tidseffektivt sätt felsöka problem som uppstår vid kodning. Genom att analysera data, kodstrukturer och felmeddelanden kan AI-modellen rekommendera tillvägagångssätt för användaren att lösa kodningsfelet. Modellen ger förslag på felsökningsstrategier och tekniker som hjälper utvecklaren att identifiera felet som uppstått (Surameery & Shakor, 2023).

3.7 Etiska dilemman, lagar och regler

Med allt fler aktiva användare av ChatGPT, i synnerhet inom arbetsrelaterade områden finns det etiska dilemman som har uppstått till följd av dess exponentiella tillväxt. Trots att AI har bevisat sig vara ett hjälpmedel som underlättat mänskliga behov i vardagen, men även automatiserat och effektiviserat arbetsuppgifter, tillkommer problematik som bör uppmärksammas. I artikeln *Paradoxes of artificial intelligence in consumer markets: Ethical challenges and opportunities,* skriver författarna Du. & Xie (2021) om etiska dilemman i samband med användning av AI.

Automatisering inom arbetsrelaterade områden har i vissa fall redan lett till bortfall av jobb. Människor har blivit ersatta av mer tidseffektiva och precisa robotar. Produktionslinor inom industribranschen har implementerat AI-drivna robotar och automationssystem, som ersatt mänskliga arbetare i tillverkningsprocesser. Inom kundtjänst har virtuella assistenter tagit över jobb som traditionellt krävde mänsklig närvaro och service. Nu kan AI-modeller istället ge grundläggande kundsupport där de svarar på FAQ (frequently asked questions) och hjälpa till med enklare transaktioner (Du & Xie, 2021).

Mängden känslig data som förses till olika AI-modeller har fått privatpersoner, likt företag att uttrycka sin oro kring vad som sker med deras data. Integritetsintrång är en av de främsta farhågorna som lyfts fram. AI-system är i många fall i behov av personuppgifter för att fungera effektivt. Om uppgifterna hanteras felaktigt, nås av obehöriga personer eller om någon obehörig person får tag i dem på grund av säkerhetsbrister, kan detta leda till att känslig information och personlig integritet äventyras (Du & Xie, 2021). Det råder oro kring hur personuppgifter kan missbrukas eller delas utan personens samtycke. Du & Xie (2021) pekar på att det finns en rädsla för att AI-system kan använda personuppgifter för ändamål utöver den ursprungliga avsikten. Såsom riktad reklam, profilering eller påverkan av beteende, utan tydligt användaresamtycke eller kontroll.

4. Empiri/Resultat

I följande kapitel ges en översiktlig beskrivning av det empiriska underlag som ligger till grund för denna studie. Empirin har samlats in genom intervjuer och observationer för att besvara studiens syfte och tillhörande frågeställning.

4.1 Inledande intervju

Intervjuerna som genomfördes gjordes utifrån ett explorativt förhållningssätt. Med löpande analyser kunde de inledande intervjuerna ge upphov till att skapa mer definierade och relevanta frågor i förhållande till studiens frågeställning. Genom att bearbeta insamlad empiri, fick skribenterna nya insikter och idéer för att skapa en bättre struktur i arbetet. Vidare har urvalet av respondenter varit baserat på olika kompetenser och erfarenheter inom mjukvaruutveckling. *Bild 4* visar vilken ålder, tjänsteställning samt programmeringsspråk respektive respondent har. Under avsnittet *bilagor* finns intervjufrågor och svar från samtliga intervjuer.
Nedan presenteras en sammanfattning av respektive intervju, där varje respondent är identifierad som Respondent A till D.

Bild 4

Respondent	Ålder	Område	Tjänsteställning	Programmeringsspråk
A	32	Webbutveckling	Senior	PHP
B	33	Mobila nätverk	Senior	Erlang
C	23	Webbutveckling	Junior	Javascript
D	32	Embedded systems	Senior	Python

4.1.1 Respondent A

Respondent A innehar en akademisk bakgrund bestående av en civilingenjörsutbildning från Chalmers tekniska högskola, med inriktning datavetenskap. Respondenten är en fullstack utvecklare och serveradministratör som primärt jobbar med PHP och JavaScript. Arbetsuppgifterna innefattar att lösa lite mer avancerade tekniska uppgifter och att ta fram CRM-lösningar åt kund. Totalt sett har respondenten över tio års erfarenhet av programmering och en spetskompetens inom webbutveckling. När respondenten idag kommenterar kod, sker det i samband med att hen jobbar inom stora projekt, där större kodstycken behöver beskrivas. Detta för att underlätta samarbetet med andra involverade i

projektet. Kommentarerna syftar till att översiktligt beskriva vad olika klasser ska hantera i systemet. Enligt Respondent A underlättar detta förvaltning av system, där kommentaren ger beskrivning för respektive klass och dess innebörd. Med tätt samarbete med kund är det också viktigt att genom kommentarer upplysa kunden om vad olika kodstycken gör. Detta sker för att ge vidare beskrivning på delar som anses vara relevanta, då systemet inte förvaltas av respondenten själv. I nuläget använder Respondent A inte något utomstående hjälpmedel eller särskilda riktlinjer till att kommentera kod, istället förhåller sig respondenten till hur utvecklingsmiljön i PHP och JavaScript tillåter hen att skriva kommentarer. Vid uppdatering av kod anser Respondenten att det är viktigt att även uppdatera tillhörande kommentarer.

Respondenten har vid tillfällen mottagit och ändrat kod på befordran av kund där kommentarer har varit bristfälliga. Exempelvis att det saknats kommentarer på längre kodstycken vilket inneburit extra arbete då respondenten tvingats gå igenom kodstycken grundligt.

Avslutningsvis ser Respondent A inte ChatGPT som varken ett hot eller något hjälpmedel, med motiveringen att man själv behöver kunna vara självgående och duktig som programmerare för att kunna nyttja tjänsten till dess fulla potential.

4.1.2 Respondent B

Respondent B arbetar som software developer och har en akademisk bakgrund med inriktning Software engineering and management på Göteborgs Universitet. Arbetsuppgifterna går ut på att producera mjukvara för telekomindustrin, i synnerhet mobila nätverk. Vidare är respondenten en backend-utvecklare, som jobbar inom programmeringsspråket Erlang. Spetskompetenser involverar att programmera i Erlang, inom kontextet av telekom och 5G. Den totala erfarenheten inom programmering för Respondent B är cirka 7 år.

När respondenten idag kommenterar kod, sker det i samband med större kodstycken där det är flera programmerare involverade i projektet. Kommentarerna är mestadels till för att översiktligt beskriva vad olika klasser ska hantera i systemet. Respondenten fortsätter med att förklara att Erlang inte är statisk och att det finns specifikationer som fungerar likt kommentarer. Deras syfte är att beskriva vilken datatyp som bör användas för en funktion och vilka outputs som tillkommer därefter. Vid användning av hjälpmedel säger sig Respondent B inte riktigt veta om några hjälpmedel kopplat till kodning. Istället lyfter respondenten fram att vissa kinesiska programmerare använder sig av google translate, utifrån hens egna upplevelser.

Vid frågan om ifall särskilda riktlinjer behöver följas för att kommentera kod, menar respondenten att det egentligen inte riktigt finns några sådana. Istället blir det att man använder sig av sunt förnuft. Vissa kommentarer behöver mer detaljer, medan andra behöver mindre beskrivningar. Strikta riktlinjer från den administrativa sidan följs däremot inte.

Respondenten fortsätter med att berätta; för att få bättre kongruens mellan olika avdelningar inom företaget är det viktigt att kommentera kod. Detta betonas ytterligare hos stora företag, där man har en större variation bland arbetarna. Det kan vara kulturella skillnader, språkbarriärer och gap i kompetens.

Respondent B besvarar frågan ifall hen har sett bristfälliga kommentarer, med att säga att hen har sett det mesta. Allt från oläsbar kod, till kod som saknar logik och kontext. Kommentarer hade i det fallet kunnat hjälpa den som granskar koden att förstå syftet åtminstone. Hens förklaring på varför detta kan uppstå är kulturella skillnader, olika standarder vad gäller kodning och språkbarriärer. Vid uppdatering av kod är respondenten noggrann med att uppdatera kommentarerna också. Avslutningsvis tror Respondent B att det finns en chans att ChatGPT skulle kunna vara ett hjälpmedel, som hade kunnat lösa de olika problemen som kan uppstå vid kommentering av kod. Respondenten betonar däremot att det finns andra dilemman som berör sekretess. Många företag vill inte använda sig av externa AI-modeller som tillhandahåller deras interna kod.

4.1.3 Respondent C

Respondent C är en tidigare systemvetarstudent på Högskolan i Borås. Innan respondenten påbörjade sina högskolestudier, läste hen en teknisk linje på gymnasiet med inriktning informationsteknik. Det inkluderade bland annat databasteknik, webbutveckling och objektorienterad programmering. Totalt har respondenten 9 års erfarenhet av programmering och ägnar sig åt fullstackutveckling, där primärt Python och JavaScript används.

Respondent C känner att kod i 90% av fallen blir självförklarande. I hög grad känner respondenten att man inte behöver skriva kommentarer, men att det finns fall där de kan användas som ett stöd. Inom olika programmeringsspråk finns det standarder som förklarar hur man ska gå tillväga för att arbeta med kommentarer. Respondenten nämner C# som ett exempel, där man ovanför alla metoder och klasser kan lägga in en viss struktur på en kommentar. Med hjälp av IntelliSense kan man i Visual Studio påpeka vad ett kodstycke gör. Vidare har man även möjligheten att definiera vad olika funktioner gör och möjligheten att lägga till en beskrivande text. Detta innebär att de som läser beskrivningen kan förstå vad en klass eller metod är till för.

Respondent C nämner att hen inte använder sig av några specifika hjälpmedel vid kommentering av kod. Däremot påpekar respondenten att Visual Studio, i synnerhet inom C# att ifall man gör tre snedstreck så får man ett template som automatiskt lägger upp ett område där man kan lägga in en summary. Den lägger automatiskt in alla parametrar och vilket returvärde man får. Vid frågan om riktlinjer inom kommentering av kod, svarar respondenten att det på basis av självaste programmeringsspråket förekommer specifikt vid dokumentation. Olika programmeringsspråk har olika sätt att kommentera på och för att linting (statisk analys på source code) ska kunna genomföras, bör man förhålla sig till respektive

programmeringsspråks riktlinjer. Respondenten betonar att utöver att följa de direktiv som enskilda programmeringsspråk har, bör kommentarerna vara korta och koncisa. De ska inte ta över koden och de bör vara relevanta och lätta att förstå.

När Respondent C blir tillfrågad om hen har upplevt dåligt kommenterad kod innan, påstår sig hen inte ha sett särskilt många kommentarer alls. Respondenten utvecklar vidare att det beror mycket på vilket sätt man jobbar på eller vilken roll man har på ett företag. Många API är exempelvis väldigt dåligt kommenterade, de beskriver vad självaste endpointen är och vad den får för return code. Men den skriver inte vilket objekt man får tillbaka. Ifall man jobbar med ett projekt över en längre period där man uppdaterar kod är det däremot viktigt att kommentera den, då fler aktörer är involverade och man behöver göra det enkelt för nya människor att kunna sätta sig in i vad det är man har kodat. Respondent C tror avslutningsvis att LLM kommer att skala upp betydligt mer än vad de redan har gjort. Respondenten fortsätter att utveckla svaret med att säga att de troligtvis kommer att ta mycket större yta än vad man tror idag. Folk underskattade internet och hen tillägger även att det fanns de som överskattade det fenomenet men att resultaten talar sitt tydliga språk idag. AI sägs vara en större utveckling än vad internet var när det kom ut och Respondent C tycker att enbart det uttalandet bör säga hur det kommer att påverka världen, oavsett vilket yrke man väljer.

4.1.4 Respondent D

Respondent D har en kandidat inom matematik och sammanfattat 4 års kurser i programmering. Utöver det har hen även några programmeringskurser från gymnasiet som bakgrund. I nuläget jobbar respondenten som mjukvaruutvecklare. De primära arbetsuppgifterna handlar om att implementera nya funktioner i mjukvaran, göra code review av andra personers kod och även bug rättningar. Spetskompetens är inom programmeringsspråket Python och specifikt inom embedded systems. Vidare innefattar tjänsten back-end utveckling för en call service. Totalt sett har respondenten 8 års erfarenhet inom programmering.

Vid kommentering av kod använder Respondent D ett digitalt verktyg vid namn Doxygen som hjälpmedel, detta för att skapa “skelettet” för funktionens kommentar. Doxygen är en dokumentationsgenerator och ett statiskt analysverktyg. Respondenten brukar skriva en längre kommentar i alla nya funktioner som tillkommer, där kommentaren är ämnad att översiktligt beskriva hur funktionen ska bete sig. Vid frågan om särskilda riktlinjer följs vid kommentering av kod, utvecklar respondenten vidare att man endast jobbar med universellt kända normer och inga särskilda riktlinjer från företaget. Kommentarer ska inte vara för långa och målet är att göra koden så självklar som möjligt med exempelvis bra namn på variabler och bra struktur. Respondenten menar även att det är när ett beteende eller konsekvens av funktionen inte är uppenbar som man ska skriva en kommentar som klargör detta.

Respondent D anser det viktigt att kommentera kod, i syfte att ge en översiktlig bild över vad koden ska göra. Annars uppstår det en situation där man först måste förstå de små detaljerna, för att sedan få en helhetsbild. Kommentarer underlättar denna process och gör att det går mycket snabbare. Respondenten delar med sig av sina erfarenheter med bristfälliga kommentarer, där hen pekar på att det sker ofta. Om koden är gammal finns det alltid risk att mycket inte är kommenterat. Det beror på att flera människor vid olika tidpunkter haft olika uppfattningar om vad som är bra kommentarer, medan vissa inte tyckt att det varit nödvändigt alls. Konsekvensen av det blir att stora delar av koden inte blir kommenterad.

Avslutningsvis ger Respondent D ett utvecklat svar på sin syn om LLM och hur det kan påverka yrket som programmerare. Respondenten menar att en pessimistisk syn är att det ger upphov till att det blir mycket enklare att programmera väl och detta gör att tröskeln att programmera blir betydligt mindre. Då det är enklare att skapa mjukvara, tror hen att det inte kommer att vara lätt att effektivt hantera produktivitets-boosten som AI medför. Detta gör att jobben kommer att bli färre över tid och att programmeringsyrket inte lär vara en brist längre. Då det kommer att vara enklare att programmera, blir följden att mjukvara överlag blir billigare. Det i sin tur gör att det blir svårare att tjäna pengar. Av den anledningen tror Respondent D att bolag som främst säljer mjukvara kommer att bli färre. Vidare pekar respondenten på att det är svårt att förutspå framtiden och att respondentens egna tankar kanske inte alls kommer att stämma.

4.2 Observationer

Efter de inledande intervjuerna blev respektive respondent ombedd att välja ut ett kodstycke på cirka 50-100 rader i sitt primära programmeringsspråk som var relativt komplext och saknade kommentarer.

För att kunna använda ChatGPT på bästa sätt just i avseende att kommentera koden, skapades och reviderades en instruktion. Syftet med instruktionen var att förklara för ChatGPT vad som skulle göras med den inmatade okommenterade koden. Instruktionen testades och reviderades av skribenterna tills önskat resultat uppnåddes. ChatGPT har vissa begränsningar när det gäller antalet tecken som kan hanteras åt gången, därför inkluderade instruktionen också en förklaring av hur ChatGPT skulle förhålla sig till input. Dessutom behövdes ytterligare information om variabler, såsom vilket programmeringsspråk som användes och vilken kodkonvention som skulle användas för att kommentera koden. Detta inkluderade också i vilken utsträckning ChatGPT skulle kommentera koden, med avseende på vad Nielebock et al. (2019) beskriver som "implementeringskommentarer" respektive "dokumentationskommentarer." Slutligen uppmanades ChatGPT också i instruktionen att endast kommentera inom koddisplayen och inte lägga till kommentarer ovanför eller nedanför.

Resultatet blev följande instruktioner, varav senaste användes under observation:

Iteration 1:
"Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: ‘x’ for single-line comments and ’y’ for multi-line comments. Note that I only want comments within the code, and not below any display outputs."

Revidering: ChatGPT kommer inte ihåg prompten för nästa stycke kod som matas in

Iteration 2:
"Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: ‘x’ for single-line comments and ‘y’ for multi-line comments. Note that I only want comments within the code, and not below any display outputs. The code is divided into smaller chunks, when you are done with the first input, await the next and do the same operation as the last"

Revidering: ChatGPT börjar kommentera senaste inmatad kod. Behöver vänta på input.

Iteration 3:
"Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: ‘x’ for single-line comments and ‘y’ for multi-line comments. Note that I only want comments within the code, and not below any display outputs. The code is divided into smaller chunks, when you are done with the first input, await the next and do the same operation as the last. Also await my input of code every time"

Nedan visas resultat för valt kodstycke av respektive respondent, kontra framställningen av ChatGPT. För att främja läsbarhet har endast ett visst stycke kod använts för respektive.
(Se fullständigt kodstycke för Javascript, respektive Python i bilagor)

4.2.1 Respondent A (PHP)

Okommenterad kod

```php
<?php

function readIO():string {
    $io = fopen ("php://stdin","r");
    return fgets($io);
}

echo "Hello and welcome to php cli ed\r\n";

while( true ) {
    echo "Ge mig en matematisk ekvation så ska jag räkna ut den åt dig..\r\
    $io = 'echo '.readIO().';';
    try {
        echo "svaret blir: ";
        $c = eval($io);
        echo "\r\n\r\n";
    } catch(Throwable $e) {
        echo "Det blev något fel med din matematiska ekvation, pröva gärna
    }
}
```

Kommenterad kod

```php
<?php

// Function to read input from the command line
function readIO():string {
    $io = fopen ("php://stdin","r");
    return fgets($io);
}

// Greet the user
echo "Hello and welcome to php cli ed\r\n";

// Loop until user terminates the program
while( true ) {

    // Ask the user for a mathematical equation to evaluate
    echo "Ge mig en matematisk ekvation så ska jag räkna ut den åt dig..\r\n

    // Read the user's input and create an expression to evaluate using eval
    $io = 'echo '.readIO().';';

    try {
        // Evaluate the expression and print the result
        echo "svaret blir: ";
        $c = eval($io);
        echo "\r\n\r\n";
    } catch(Throwable $e) {
        // If an error occurs, catch it and inform the user
        echo "Det blev något fel med din matematiska ekvation, pröva gärna i
    }
}
```

4.2.2 Respondent B (ERLANG)

Okommenterad kod

```
setopts_active(#state{socket=Socket, transport=Transport, opts=Opts}) ->
    N = maps:get(active_n, Opts, 100),
    Transport:setopts(Socket, [{active, N}]).

loop(State=#state{parent=Parent, socket=Socket, transport=Transport,
        opts=Opts, timer=TimerRef, children=Children}, Buffer) ->
    Messages = Transport:messages(),
    InactivityTimeout = maps:get(inactivity_timeout, Opts, 300000),
    receive

        {OK, Socket, Data} when OK =:= element(1, Messages) ->
            parse(set_idle_timeout(State), << Buffer/binary, Data/binary >>)
        {Closed, Socket} when Closed =:= element(2, Messages) ->
            Reason = case State#state.http2_status of
                closing -> {stop, closed, 'The client is going away.'};
                _ -> {socket_error, closed, 'The socket has been closed.'}
            end,
            terminate(State, Reason);
        {Error, Socket, Reason} when Error =:= element(3, Messages) ->
            terminate(State, {socket_error, Reason, 'An error has occurred o
        {Passive, Socket} when Passive =:= element(4, Messages);

                Passive =:= tcp_passive; Passive =:= ssl_passive ->
            setopts_active(State),
            loop(State, Buffer);
```

Kommenterad kod

```
%% Set the active option for the socket
setopts_active(#state{socket=Socket, transport=Transport, opts=Opts}) ->
    N = maps:get(active_n, Opts, 100),
    Transport:setopts(Socket, [{active, N}]).

%% The main loop for the Cowboy server
loop(State=#state{parent=Parent, socket=Socket, transport=Transport,
        opts=Opts, timer=TimerRef, children=Children}, Buffer) ->
    Messages = Transport:messages(),
    InactivityTimeout = maps:get(inactivity_timeout, Opts, 300000),

    %% Wait for incoming messages and act accordingly
    receive
        {OK, Socket, Data} when OK =:= element(1, Messages) ->
            parse(set_idle_timeout(State), << Buffer/binary, Data/binary >>)
        {Closed, Socket} when Closed =:= element(2, Messages) ->
            Reason = case State#state.http2_status of
                closing -> {stop, closed, 'The client is going away.'};
                _ -> {socket_error, closed, 'The socket has been closed.'}
            end,
            terminate(State, Reason);
        {Error, Socket, Reason} when Error =:= element(3, Messages) ->
            terminate(State, {socket_error, Reason, 'An error has occurred o
        {Passive, Socket} when Passive =:= element(4, Messages);
            Passive =:= tcp_passive; Passive =:= ssl_passive ->
            setopts_active(State),
            loop(State, Buffer);
```

4.2.3 Respondent C (Javascript)

Okommenterad kod

```
window.addEventListener('mouseup', function() { mouseIsDown = false; });

function neutralizeColumn(columnElement) {
    let classes = ['obstacle', 'start', 'goal'];

    for (let i = 0; i < classes.length; i++) {
        columnElement.classList.remove(classes[i]);
    }
}

function updateObstacle(columnElement) {
    let isObstacle = columnElement.classList.contains('obstacle');
    neutralizeColumn(columnElement);
    if (!isObstacle) {
        columnElement.classList.add('obstacle');
    }
}

function updateStart(columnElement) {
    neutralizeColumn(columnElement);
    let starts = document.querySelectorAll('.start');
    for (let i = 0; i < starts.length; i++) {
        neutralizeColumn(starts[i]);
    }
    columnElement.classList.add('start');
}

function updateGoal(columnElement) {
    neutralizeColumn(columnElement);
    let starts = document.querySelectorAll('.goal');
    for (let i = 0; i < starts.length; i++) {
        neutralizeColumn(starts[i]);
    }
    columnElement.classList.add('goal');
}
```

Kommenterad kod

```
// This function sets mouseIsDown to false when the mouse button is released
window.addEventListener('mouseup', function() { mouseIsDown = false; });

// This function removes the obstacle, start, and goal classes from a given
function neutralizeColumn(columnElement) {
    let classes = ['obstacle', 'start', 'goal'];

    // Loop through each class and remove it from the element's class list.
    for (let i = 0; i < classes.length; i++) {
        columnElement.classList.remove(classes[i]);
    }
}

// This function updates the given column element as an obstacle.
function updateObstacle(columnElement) {
    // Check if the element is already an obstacle.
    let isObstacle = columnElement.classList.contains('obstacle');

    // Remove all existing classes from the element.
    neutralizeColumn(columnElement);

    // If the element wasn't already an obstacle, add the 'obstacle' class.
    if (!isObstacle) {
        columnElement.classList.add('obstacle');
    }
}
```

4.2.4 Respondent D (Python)

Okommenterad kod

```python
import machine
import time

SENSOR_PIN = 4
ACTUATOR_PIN = 5
THRESHOLD = 500

sensor = machine.ADC(SENSOR_PIN)
actuator = machine.Pin(ACTUATOR_PIN, machine.Pin.OUT)

sensor_history = []

def average_sensor_value():
    if len(sensor_history) > 0:
        return sum(sensor_history) / len(sensor_history)
    else:
        return 0

def process_sensor_value(value):
    if value > THRESHOLD:
        actuator.on()
        time.sleep_ms(500)
        actuator.off()
    else:
        actuator.off()

while True:
    sensor_value = sensor.read()

    sensor_history.append(sensor_value)

    if len(sensor_history) > 10:
        sensor_history.pop(0)

    average_value = average_sensor_value()

    process_sensor_value(average_value)

    time.sleep(1)
```

Kommenterad kod

```
SENSOR_PIN = 4
ACTUATOR_PIN = 5
THRESHOLD = 500

# Initialize ADC pin for sensor and GPIO pin for actuator
sensor = machine.ADC(SENSOR_PIN)
actuator = machine.Pin(ACTUATOR_PIN, machine.Pin.OUT)

# List to store previous sensor values
sensor_history = []

def average_sensor_value():
    """Calculate the average sensor value from the history"""
    if len(sensor_history) > 0:
        return sum(sensor_history) / len(sensor_history)
    else:
        return 0

def process_sensor_value(value):
    """Perform action based on sensor value"""
    if value > THRESHOLD:
        # If sensor value is above threshold, turn on the actuator for 500 m
        actuator.on()
        time.sleep_ms(500)
        actuator.off()
    else:
        # If sensor value is below threshold, keep the actuator off
        actuator.off()

while True:
    # Read sensor value
    sensor_value = sensor.read()

    # Add current sensor value to history
    sensor_history.append(sensor_value)

    # Keep history size limited to 10 values
    if len(sensor_history) > 10:
        sensor_history.pop(0)

    # Calculate average sensor value
    average_value = average_sensor_value()

    # Process the average sensor value to determine the action
    process_sensor_value(average_value)

    # Delay before next iteration
    time.sleep(1)
```

4.3 Avslutande intervjuer

En avslutande intervju genomförs med respektive respondent där bland annat resultat av ChatGPTs framställning diskuteras. Intervjun genomfördes parallellt med observationen, då respondenterna analyserade ChatGPTs output.

4.3.1 Respondent A

Respondenten anser att ChatGPTs kommentarer generellt sett var korrekta och överensstämmer med koden. Respondenten anser att färre kommentarer skulle ha varit tillräckligt, med korta beskrivningar av varje funktion. Hen anser att kommentera kod är en enkel och intuitiv process och ser inte behovet av ett verktyg för att underlätta detta.

Respondenten ser inte relevansen av att använda ChatGPT för att kommentera kod. Respondenten tror att det skulle vara svårt för verktyget att förstå hur man kommenterar ett större system när man bara får kodsnuttar. Respondenten tycker också att det är tråkigt att kopiera och klistra in kod i delar för att producera kommentarer. Personligen känner respondenten inte behovet av ett verktyg för att kommentera kod eftersom det inte tar mycket tid att göra det manuellt.

Respondenten anser att användbarheten hos ett verktyg för kommentering av kod beror på vilken typ av kommentar det handlar om. Respondenten ser potential i verktyg som beskriver klasser, medlemsfunktioner och publika variabler, samt förklarar datatyper som accepteras som argument. Dock kan specifika funktioner kräva personliga kommentarer som endast programmeraren kan tillhandahålla. Respondenten tror att automatisering kan ha en positiv inverkan vid uppdatering av kod genom att spara tid på manuell kommentering av varje kodsnutt.

Respondenten anser att det är acceptabelt att kommentera kod för att ge en översikt, men tror att fokus bör ligga mer på att skapa självförklarande kod än omfattande kommentarer. De tror att ChatGPT på sikt kan hjälpa till att motverka mänskliga fel i kodkommentarer, särskilt om det används brett. Många fel vid kommentering av kod idag beror ofta på att kommentering ses som tråkig eller onödig. Respondenten tror att ChatGPT kan hjälpa till att minska dessa fel. Vidare menar respondenten att ChatGPT potentiellt skulle kunna generalisera kommentarer för att göra dem begripliga för alla inblandade.

Respondenten tror att det skulle vara möjligt att skapa ett API framöver som integrerar ChatGPT som ett verktyg i olika kodredigerare (IDE).

Respondenten uttrycker viss oro över att förlita sig för mycket på verktyg som ChatGPT eftersom det kan leda till minskad kritiskt tänkande, kreativitet och självständig problemlösning. De anser att individer behöver vara kompetenta inom sitt yrke för att kunna utnyttja ChatGPTs potential fullt ut. Medan mindre erfarna programmerare kan dra nytta av

verktyget kan erfarna utvecklare vara tveksamma till att främst förlita sig på ChatGPT som sin huvudsakliga metod. Dock erkänner respondenten att ChatGPT kan vara till hjälp för mindre erfarna kodare som kan ha nytta av att använda ChatGPT istället för att söka hjälp från erfarna kollegor eller genom att söka online.

4.3.2 Respondent B

I intervjun uttrycker respondenten åsikter och synpunkter angående användningen av ChatGPT för kommentering av kod. Respondenten ansåg att de genererade kommentarerna var vagt formulerade och saknade egentlig värde för att förstå koden. Det framhölls att det inte fanns någon tydlig nytta med att använda ChatGPT för kommentering av kod i det specifika fallet. Respondenten betonade vikten av meningsfull och lättförståelig kommentering av kod, och påpekade att det är en viktig del av utvecklingsprocessen.

Trots detta var respondenten öppen för användningen av automatiserade verktyg för kommentering av kod, särskilt för att främja enhetlighet i större utvecklingsteam. Framförallt eftersom respondenten varit med om dålig dokumentation i källkod. Det betonades dock att befintliga lösningar, som ChatGPT, behöver förbättras för att vara mer användbara och praktiska i verkliga scenarier. Respondenten ansåg att automatisering av kommentering av kod kunde vara fördelaktig i vissa situationer för att uppnå standardiserade tillvägagångssätt och undvika individuella skillnader i kod- och dokumentationsstilen.

När det gäller att automatisera uppdatering av kod var respondenten positiv och betonade att verktyget måste fungera bra och vara tillförlitligt för att vara användbart. Respondenten såg en potential i verktyg som kan analysera och identifiera nödvändiga kodändringar, vilket skulle spara tid och underlätta förvaltning.

Respondenten var öppen för användningen av verktyg som ChatGPT, men underströk att kritiskt tänkande och självständig problemlösning är viktiga aspekter som inte bör kompromettetas. Respondenten ansåg inte att människor skulle förlita sig för mycket på verktyg om det inte effektivt minskar tid och kostnader. Det poängterades att verktyg inte bör ersätta dessa viktiga förmågor utan snarare stödja och underlätta arbetsprocessen.

Respondenten betraktade inte stora språkmodeller (LLMs) som nödvändiga för att vara konkurrenskraftig i dagsläget. De insåg dock potentialen för tids- och kostnadsbesparingar om LLMs integrerades väl och höll hög kvalitet. Respondenten var öppen för att verktyg som ChatGPT kunde bli en integrerad del av programmeringsyrket i framtiden, förutsatt att de förbättras och tydliga metoder för integration etableras.

4.3.3 Respondent C

Enligt respondenten är majoriteten av kommentarerna överflödiga och tillför inte något värdefullt till förståelsen av själva koden. Kommentarer bör enligt Respondent C istället vara meningsfulla och ha en faktisk betydelse. Respondenten anser att en sista kommentar som beskriver hela funktionen skulle vara tillräcklig, istället för att ha flera kommentarer som inte tillför något. Vidare menar respondenten att när allt kommenteras förlorar kommentarerna sitt värde. Syftet med kommentarer är att tillföra något vid läsning, och om projektet kräver en sådan mängd kommentarer bör man istället se över projektets strukturer. Kommentarer bör vara relevanta och inte vara lika långa eller längre än koden själv.

Respondenten uttrycker dock intresse för att använda ett verktyg för kommentering av kod i framtiden. Hen anser att verktyg som kan generera relevanta och informativa kommentarer skulle vara till stor nytta för att förbättra kodens läsbarhet och förståelse. Automatisering av kommenteringen av kod ses därmed positivt av respondenten, då det kan spara tid och förbättra enhetligheten i kommentarerna. Respondenten betonar däremot vikten av att ha kontroll över automatiseringen, i syfte att säkerställa att kommentarerna är korrekta och relevanta.

Respondenten utvecklar sina tankar med att säga att ChatGPT kan vara till hjälp för att motverka mänskliga misstag i koddokumentation. Olika verktyg skulle eventuellt kunna ge förslag på kommentarer som är mer informativa och fullständiga än vad enbart en människa kan skriva. Detta skulle enligt respondenten kunna bidra till att förbättra kvaliteten på kommentarerna och minska risken för ofullständiga eller saknade kommentarer. Just vad gäller uppdatering av befintlig källkod anser respondenten att verktyg skulle kunna nyttjas för att säkerställa att även kommentarer uppdateras i samma veva. Respondenten påpekade även att man genom standardisering skulle kunna erhålla mer objektiva och enhetliga förslag på kodändringar, vilket kan skapa mer konsistent kod och undvika individuella stilpreferenser.

Respondenten anser inte att människor blint kommer att förlita sig på verktyg som ChatGPT för att kommentera kod. Avslutningsvis har respondenten inga åsikter kring huruvida Large Language Models (LLMs), som ChatGPT, kommer att bli nödvändiga för att vara konkurrenskraftig i framtiden.

4.3.4 Respondent D

Enligt respondenten, var ChatGPTs förmåga att kommentera kod varierande. Ibland gav ChatGpt relevanta och användbara kommentarer, men kunde också missa viktiga aspekter eller ge irrelevanta och redundanta kommentarer. Respondenten belyser försiktighet med att förlita sig helt på ett AI verktyg för att kommentera kod, eftersom det ännu inte är perfekt och kan ge felaktiga rekommendationer. Sådana felaktigheter skulle kunna leda till att kod istället blir svårläst.

Däremot kunde respondenten se vissa fördelar med att använda ett verktyg för kommentering av kod i framtiden. Det skulle kunna underlätta och påskynda utvecklingsprocessen i vissa avseenden, särskilt i stora och komplexa projekt. Respondenten betonade dock vikten av att komplettera verktyget med människors insikt och bedömning för att säkerställa att kommentarerna är korrekta och meningsfulla.

När det gäller automatisering av kommentering av kod, ansåg respondenten att det kunde vara fördelaktigt i vissa situationer, särskilt för att generera grundläggande dokumentation eller förklaringar för rutinmässiga kodmönster. Samtidigt betonade respondenten att vissa kommentarer kräver mänsklig insikt och förståelse för kontexten, vilket kan vara svårt att helt automatisera.

Respondenten ansåg att ChatGPT och liknande verktyg skulle kunna motverka mänskliga misstag vad gäller kommentering av kod genom att identifiera och rätta till ofullständiga kodavsnitt med bristfälliga och/eller avsaknad av kommentarer. Det skulle därmed kunna fungera som ett komplement till mänsklig granskning och förbättra kodens läsbarhet och underhållbarhet.

När det gäller att automatisera uppdatering av kod, ansåg respondenten att ChatGPT kunde vara till hjälp genom att generera förslag eller ge vägledning för att uppdatera befintlig kod baserat på ändringar eller nya krav. Det skulle kunna bidra till enhetlighet inom projekt och att subjektiva olikheter mellan utvecklare undviks. Samtidigt ansåg respondenten att människor kommer fortsätta att spela en aktiv roll vid bedömning och godkännande av föreslagna ändringarna.

När det gäller användningen av verktyg som ChatGPT, trodde respondenten att det fanns en risk att mjukvaruutvecklare förlitar sig för mycket på dem, vilket kan reducera kritiskt tänkande. Det är enligt respondenten viktigt att använda olika verktyg som komplement, och inte blint förlita sig helt på dem.

Enligt respondentens åsikt kan LLMs, såsom ChatGPT, vara en tillgång för företag som strävar efter att vara konkurrenskraftiga inom programmeringsområdet. Genom att automatisera vissa uppgifter, öka produktiviteten och erbjuda snabbare svar på utvecklingsfrågor kan dessa modeller bidra till att förbättra arbetsflöden. Trots detta understryker respondenten vikten av att upprätthålla mänsklig expertis vid hanteringen av komplexa problem, eftersom LLMs inte kan betraktas som en universell lösning för all programmering. Det är avgörande att använda LLMs och mänsklig kunskap på ett balanserat sätt för att uppnå optimala resultat.

5. Analys

I kapitlet besvaras studiens forskningsfråga genom att jämföra den analyserade datan med det teoretiska ramverket, vilket sedan generaliseras till en övergripande nivå.

5.1 Inledande intervju

5.1.1 Tjänsteställning/erfarenheter

Respondent A har en civilingenjörsutbildning inom datavetenskap och har över tio års erfarenhet av programmering. Respondenten är en fullstack utvecklare och serveradministratör med fokus på PHP och JavaScript. Spetskompetensen ligger inom webbutveckling.

Respondent B har en akademisk bakgrund inom Software Engineering and Management och har cirka sju års erfarenhet av programmering. Respondenten arbetar som en backend-utvecklare inom telekomindustrin och är specialiserad på programmeringsspråket Erlang, särskilt inom kontexten av telekom och 5G.

Respondent C har en tidigare systemvetarutbildning och har ägnat sig åt fullstack utveckling med huvudsaklig användning av Python och JavaScript. Respondenten har 9 års erfarenhet av programmering och har bredare kunskap inom webbutveckling.

Respondent D har en kandidatexamen i matematik och har cirka 8 års erfarenhet av programmering. Respondenten arbetar som mjukvaruutvecklare med specialisering inom programmeringsspråket Python och embedded systems. Nuvarande tjänst involverar backend-utveckling för en call service.

Liknelser:

- Samtliga respondenter har en akademisk bakgrund inom IT-relaterade områden.
- Alla har erfarenhet av programmering och är aktiva inom mjukvaruutveckling.
- De har olika specialiseringar inom olika programmeringsspråk och tekniska områden.

Skillnader:

- Respondent A har den längsta erfarenheten av programmering med över tio år.
- Respondent B är specialiserad på Erlang och arbetar inom telekomindustrin, medan Respondent C verkar ha en bredare kunskap inom webbutveckling och specifikt användning av Javascript.
- Respondent A och B har en mer specifik inriktning inom sina områden (webbutveckling respektive telekom), medan Respondent C verkar ha en mer allmän kunskap inom fullstackutveckling och respondent D är specialiserad inom embedded systems.

- Respondent A har en civilingenjörsutbildning från Chalmers tekniska högskola, Respondent B har en akademisk bakgrund från Göteborgs universitet, Respondent C är en tidigare systemvetarstudent från Högskolan i Borås och Respondent D har en kandidatexamen i matematik utan specifik nämnvärd universitets tillhörighet.

Sammanfattningsvis har respondenterna olika bakgrunder, erfarenheter och specialiseringar inom programmering. De har alla en formell utbildning inom IT-området och arbetar som mjukvaruutvecklare, men deras kunskaper och kompetenser skiljer sig åt beroende på deras utbildningar och erfarenheter. Det kan dock anses att deras bakgrund inom programmering ger befogenhet att svara på IT-relaterade frågor.

5.1.2 Riktlinjer vid kommentering av kod

Respondent A använder sig inte av några särskilda riktlinjer för att kommentera kod. Respondenten följer istället utvecklingsmiljöerna i PHP och JavaScript för att bestämma hur de ska skriva kommentarer.

Respondent B anser att det inte finns några särskilda riktlinjer för att kommentera kod. Istället förlitar de sig på sunt förnuft när de väljer att kommentera.

Respondent C förklarar att olika programmeringsspråk har standarder för att arbeta med kommentarer. Respondenten betonar vikten av korta och koncisa kommentarer som inte tar över koden. Kommentarerna bör vara relevanta och lätta att förstå, särskilt vid dokumentation.

Respondent D nämner att de följer universellt kända normer vid kommentering av kod och att det inte finns några särskilda riktlinjer från deras företag. Respondenten strävar efter att göra koden så självklar som möjligt med tydliga variabelnamn och en bra struktur. Kommentarerna ska inte vara för långa.

Liknelser:

- Samtliga respondenter betonar vikten av att kommentarer ska vara relevanta och lätta att förstå.
- Ingen av respondenterna nämner att det finns strikta riktlinjer som måste följas vid kommentering av kod.
- Alla respondenterna framhåller behovet av att ha korta och koncisa kommentarer.

Skillnader:

- Respondent A förlitar sig på utvecklingsmiljöerna i PHP och JavaScript för att bestämma stil och struktur på kommentarer, medan Respondent B använder sunt förnuft.

- Respondent C nämner att det finns standarder inom olika programmeringsspråk för att arbeta med kommentarer, särskilt vid dokumentation.
- Respondent D nämner att de inte har särskilda riktlinjer från företaget utan följer universellt kända normer och fokuserar på att göra koden självklar genom variabelnamn och struktur.

Sammanfattningsvis har respondenterna olika tillvägagångssätt när det gäller kommentering av kod. Både Nielebock et al (2019) och Huang et al (2020) påpekar att riktlinjer för att både koda och kommentera ofta uppkommer från personliga preferenser och erfarenheter.
Vissa förlitar sig på utvecklingsmiljöer eller sunt förnuft, medan andra nämner standarder och dokumentationskrav inom programmeringsspråken. Trots skillnaderna betonar samtliga respondenter vikten av korta, relevanta och lättförståeliga kommentarer i koden. Huang et al (2020) belyser även vikten av att kommentarer ska användas sparsamt och endast där det förekommer komplex kod eller logik som kräver ytterligare förtydligande. Enligt Huang et al (2019) och Nielebock et al (2019) betraktas kommentering av kod som en viktig men ofta förbisedd aktivitet inom mjukvaruutveckling. Trots formella och automatiska tillvägagångssätt finns det brister i dagens praxis, särskilt när det gäller kodkonventioner (Huang et al, 2019).

5.1.3 Hjälpmedel vid kommentering av kod

Respondent A använder för närvarande inte några utomstående hjälpmedel vid kommentering av kod.

Respondent B är inte medveten om några specifika hjälpmedel kopplade till kodning. Respondenten nämner dock att vissa kinesiska programmerare använder sig av Google Translate, baserat på deras egna erfarenheter.

Respondent C använder inga specifika hjälpmedel vid kommentering av kod.

Respondent D använder ett specifikt digitalt verktyg vid namn Doxygen som hjälpmedel för att skapa "skelettet" för funktionens kommentar. Doxygen är en dokumentationsgenerator och ett statiskt analysverktyg.

Liknelser:

- Respondent A och C nämner båda att de inte använder några specifika hjälpmedel vid kommentering av kod.
- Respondent B och D nämner båda användningen av hjälpmedel, även om de använder olika typer av verktyg.

Skillnader:

- Respondent A använder inga utomstående hjälpmedel alls, medan Respondent B nämner möjligheten att vissa kinesiska programmerare använder Google Translate.

- Respondent D använder det specifika verktyget Doxygen för att underlätta skapandet av kommentarstrukturer, medan de andra respondenterna inte nämner något liknande verktyg.

Sammanfattningsvis varierar respondenternas användning av hjälpmedel vid kommentering av kod. Majoriteten använder inga utomstående hjälpmedel alls, medan andra nämner möjligheten att använda Google Translate vid eventuella språkbarriärer eller specifika verktyg som Doxygen. Det finns både likheter och skillnader mellan respondenternas tillvägagångssätt, men det är tydligt att vissa programmerare kan dra nytta av hjälpmedel för att underlätta kommentering av kod. Haleem et al (2023) skriver om hur ChatGPT kan användas som ett hjälpmedel i syfte att kommentera kod. Genom att analysera programmeringsspråket, kodstruktur och funktionskrav kan AI-modellen rekommendera bästa praxis och standarder inom koddokumentation.

5.1.4 Innebörd av att kommentera kod

Respondent A kommenterar kod i samband med stora projekt och fokuserar på att beskriva större kodstycken. Syftet är att underlätta samarbetet med andra projektmedlemmar och ge en översiktlig beskrivning av vad olika klasser ska hantera i systemet.

Respondent B kommenterar också kod i samband med stora kodstycken där flera programmerare är involverade i projektet. Kommentarerna har liknande syfte som respondent A, nämligen att ge en översiktlig beskrivning av vad olika klasser ska hantera i systemet.

Respondent C anser att kommentering av kod inte alltid är nödvändig, men det finns situationer där kommentarer kan vara till hjälp. Respondenten nämner möjligheten att definiera vad olika funktioner gör och att lägga till beskrivande text. Kommentarer blir viktiga i projekt som sträcker sig över en längre tid och när flera personer är involverade för att underlätta förståelsen för nya personer som ansluter till projektet.

Respondent D anser att det är viktigt att kommentera kod för att ge en övergripande bild av vad koden ska göra. Kommentarerna hjälper till att snabbare förstå helhetsbilden istället för att behöva förstå de små detaljerna först.

Liknelser:

- Samtliga respondenter nämner vikten av kommentarer för att ge en översiktlig beskrivning av vad koden ska göra.
- Respondent A och B kommenterar kod i samband med större kodstycken där flera programmerare är involverade i projektet.

Skillnader:

- Respondent C nämner att de inte anser att kommentering alltid är nödvändig men att det finns fall där det kan användas som stöd.
- Respondent D betonar behovet av kommentarer för att snabbare få en helhetsbild av koden.

Sammanfattningsvis har respondenterna gemensamma åsikter om att kommentarer är viktiga för att ge en översiktlig beskrivning av kodens funktion. Enligt Laramee (2013) utgör kodkonventioner, inklusive kommentering av kod, en solid grund för framgångsrik mjukvaruutveckling. Eftersom mjukvaruutvecklare spenderar en betydande del av sin tid på att underhålla, förstå och bli bekanta med befintlig källkod, är underhållsfasen, som innefattar felsökning och uppdateringar, vanligtvis den mest kostsamma fasen inom mjukvaruutveckling. Respondenterna betonar även vikten av att underlätta förståelsen för andra projektmedlemmar eller nya personer som ansluter till projektet. Enligt Nielebock et al (2019) kan ökad förståelse för programkod reducera tiden som krävs för underhåll och minska risken att införa nya buggar. Enligt Smit et al (2011) har välformulerade kommentarer en betydande inverkan genom att underlätta för nya utvecklare att förstå befintlig källkod och främja samstämmighet inom organisationer, team och användningen av olika API. Trots likheterna finns det några skillnader, särskilt kring synen på behovet av kommentering och fokus på att snabbt förstå helhetsbilden. Respondenternas åsikter kring innebörden av att kommentera kod stämmer väl överens med vad teorin beskriver.

5.1.5 Komplikationer och brister vid kommentering

Respondent A har erfarenhet av att förvalta och ändra kod från kunder där kommentarerna har varit bristfälliga. Detta har inneburit extra arbete eftersom respondenten har behövt granska kodstycken noggrant på grund av bristen på kommentarer.

Respondent B har sett många fall av bristfälliga kommentarer i koden, inklusive oläsbar kod och kod som saknar logik och kontext. Kommentarer hade kunnat hjälpa till att förstå syftet med koden och underlätta granskningen. Respondenten pekar på att kulturella skillnader, olika kodningsstandarder och språkbarriärer kan vara orsaker till bristfälliga kommentarer.

Respondent C har inte sett många kommentarer alls i sin erfarenhet. Respondenten nämner särskilt att API:er ofta är dåligt kommenterade, där de beskriver endpointen, men inte ger information om det objekt som returneras. Respondenten påpekar att det kan vara beroende av arbetsmetoder och roll på företaget.

Respondent D har upplevt bristfälliga kommentarer regelbundet, särskilt i äldre kod. Det beror på att olika personer vid olika tidpunkter har haft olika uppfattningar om vikten av kommentarer, vilket har resulterat i att stora delar av koden inte är kommenterad.

Liknelser:

- Samtliga respondenter har erfarenhet av bristfälliga kommentarer i koden och hur det kan påverka arbetet med att förstå och ändra koden.
- Alla respondenterna anser att kommentarer är viktiga för att underlätta förståelsen av koden, både för dem själva och andra som ska granska eller arbeta med koden.

Skillnader:

- Respondent B nämner kulturella skillnader och språkbarriärer som en orsak till bristfälliga kommentarer.
- Respondent C påpekar att brister vid kommentering av kod kan vara beroende av arbetsmetoder och roll på företaget.
- Respondent D pekar på att olika uppfattningar om vikten av kommentarer över tid har resulterat i bristfälliga kommentarer i äldre kod.

Sammanfattningsvis har respondenterna gemensamma erfarenheter av bristfälliga kommentarer i källkod och en gemensam uppfattning om att kommentarer är viktiga för att underlätta förståelsen av koden. De pekar dock på några skillnader, såsom kulturella skillnader, språkbarriärer, arbetsmetoder och olika uppfattningar om betydelse för kommentarer över tid. Stora projekt involverar ofta samarbete mellan flera utvecklare under längre tidsperioder. En av de största utmaningarna, enligt Laramee (2013), är när källkod skrivs utan att följa konventioner, vilket minskar återanvändbarheten och leder till onödig omskapande av kod. Otydligt skriven kod är tyvärr vanligt förekommande bland olika utvecklare och kan leda till svårigheter att förstå koden, även för den ursprungliga utvecklaren. Bristande kommentarer eller föråldrade kommentarer försvårar även uppdatering av befintlig kod (Laramee, 2013). Projekt kan även påverkas av tidsbrist och låg motivation, som Nielebock et al. (2019) betonar. Eller rent av som Salviulo & Scanniello (2014) beskriver, att professionella programmerare ofta ignorerar kommentarer i källkod. Därmed har kommentarer större betydelse för nybörjare som kan använda den extra informationen för att öka förståelsen för olika program. Enligt Laramee (2013) utgör kodkonventioner, inklusive kommentarer, en grundläggande faktor för framgångsrik mjukvaruutveckling.

5.1.6 Large language models

Respondent A ser varken ChatGPT som ett hot eller ett hjälpmedel. Respondenten anser att man som programmerare behöver vara självgående och kompetent för att kunna dra full nytta av tjänsten.

Respondent B tror att ChatGPT kan vara ett hjälpmedel och lösa problem som kan uppstå vid kommentering av kod. Respondenten påpekar dock att det finns andra dilemman, som sekretess, eftersom företag kanske inte vill använda externa AI-modeller för sin interna kod.

Respondent C tror att AI, inklusive ChatGPT, kommer att växa och ha en stor påverkan. Respondenten jämför det med internets utveckling och menar att det kommer att påverka världen oavsett yrkesval.

Respondent D har en mer pessimistisk syn på ChatGPT:s påverkan. Respondenten tror att det kommer att bli enklare att programmera med AI, vilket kan leda till färre jobbmöjligheter och en minskad efterfrågan på mjukvaror. Respondenten förutspår att företag som huvudsakligen säljer mjukvara kommer att bli färre. Respondenten belyser dock att det endast är en subjektiv bedömning utan direkt underlag.

Liknelser:

- Både Respondent A och Respondent B diskuterar potentialen i att använda ChatGPT som ett hjälpmedel för att underlätta kommentering av kod.
- Respondent C och Respondent D uttrycker båda en tro på att AI, inklusive ChatGPT, kommer att ha en betydande inverkan på världen och yrken.

Skillnader:

- Respondent A ser inte ChatGPT som vare sig hotfullt eller användbart, medan Respondent B ser en potential i att använda det som hjälpmedel, men också påpekar dilemman som sekretess.
- Respondent C fokuserar på AI:s bredare påverkan på världen och betonar dess storlek och tillväxtpotential, medan Respondent D har en mer negativ syn på de möjliga konsekvenserna av AI-utvecklingen, särskilt när det gäller jobbtillgång och företagsöverlevnad.

Sammanfattningsvis har respondenterna olika uppfattningar om ChatGPT:s potential och dess påverkan. Vissa ser det som ett möjligt hjälpmedel medan andra uttrycker farhågor och negativa konsekvenser. Det finns också skillnader i hur de betraktar AI:s bredare påverkan på världen och yrkeslivet.

5.2 Observation

Alla observationer genomfördes med hjälp av den framställda instruktionen som skribenterna skapade (beskrivs i avsnitt 4.2). Överlag lyckades ChatGPT kommentera respondenternas kod enligt de generella riktlinjer som förekommer för respektive språk. Det fanns tydliga diskrepanser mellan de olika språken, framförallt vad gäller Javascript. Javascript erhöll betydligt mer kommentarer i källkoden jämfört med ERLANG, som knappt hade några kommentarer alls. Enligt Statista (2022) placeras Javascript på 65 %, medan ERLANG ligger på under 3.5 %, och är inte ens med i listan. Resultatet kan ge en indikation på att popularitet för programmeringsspråk är en avgörande faktor för ChatGPTs kapacitet att analysera och kommentera respektive programmeringsspråk. OpenAIs senaste modell, GPT-4, tränades med en aldrig tidigare skådad mängd beräkningskraft och data (Bubeck et al, 2023), men det finns ingen information kring varifrån ChatGPT hämtat data inom programmering.

För att genomföra observation krävdes att respondenterna var fullt medvetna och insatta i den utvalda koden, för att på ett trovärdigt sätt kunna genomföra analys, samt att validera ChatGPTs output i förhållande till kodens faktiska innebörd. Instruktionen som användes var skapad för att bland annat hantera ChatGPTs begränsning på 2048 karaktärer per input. Innebörden av detta är att källkod delades upp i mindre delar och det är därmed svårt att avgöra ifall ChatGPT har förmåga att analysera hela kontextet. Detta leder till att kommentering av kod eventuellt inte utförts med hela kontextet i beaktande, vilket innebär att vissa kommentarer är felaktiga genom avsaknad av kontext.

5.3 Avslutande intervju

5.3.1 Respondent A (PHP)

ChatGPT som ett verktyg

Respondenten anser att ChatGPT skulle kunna vara ett verktyg för att kommentera kod. Respondenten tycker dock att kommentera kod är en enkel och intuitiv process och ser inte behovet av ett verktyg för att underlätta detta. Samtidigt tror respondenten att ett verktyg som ChatGPT kan ha potential att beskriva klasser, medlemsfunktioner, publika variabler och förklara datatyper. Respondenten ser fördelar med automatisering för att spara tid vid uppdatering av kod och för att undvika mänskliga fel i kodkommentarer. Respondenten tror också att ChatGPT kan generalisera kommentarer för att göra dem begripliga för alla inblandade. Slutligen nämner respondenten möjligheten att integrera ChatGPT som ett verktyg i olika kodredigerare genom ett API.

Hur presterar ChatGPT i nuläget

Enligt respondenten stämde kommentarerna väl överens med källkodens innebörd. Respondenten ser inte relevansen av att använda ChatGPT för att kommentera kod och tror att det skulle vara svårt för verktyget att förstå hur man kommenterar ett större system när man bara kan skicka in kodsnuttar i ChatGPT. Respondenten uttrycker också att hen inte känner behovet av ett verktyg för att kommentera kod eftersom det inte tar mycket tid att göra det manuellt.

Känslor kring ChatGPT och att automatisera kommentering av kod

Respondenten har blandade känslor kring att automatisera kommentering av kod med hjälp av ChatGPT. Å ena sidan tror respondenten att automatisering kan ha en positiv inverkan genom att spara tid och minska fel. De ser potential i att använda verktyg för att beskriva klasser, funktioner och datatyper. Å andra sidan uttrycker respondenten oro över att förlita sig för mycket på ett verktyg som ChatGPT, vilket kan leda till minskad kritiskt tänkande, kreativitet och självständig problemlösning. Vidare betonar respondenten som hen i tidigare intervju förklarat, att individer behöver vara kompetenta inom sitt yrke för att utnyttja ChatGPTs potential fullt ut. Respondenten anser dock att ChatGPT kan vara till hjälp för mindre erfarna mjukvaruutvecklare som kan ha nytta av att använda verktyget som alternativ, istället för att söka hjälp från erfarna kollegor eller genom att söka online.

5.3.2 Respondent B (ERLANG)

ChatGPT som ett verktyg

Respondenten anser att kommentarerna från ChatGPT var vagt formulerade och poänglösa. De tillförde inte något värde eller extra information som kunde vara till hjälp för att förstå koden. Exempelvis ansågs kommentaren "%% Handling generic calls" vara onödig och inte bidra med något användbart.

Hur presterar ChatGPT i nuläget

Respondenten var inte imponerad av ChatGPT:s prestation när det kom till att kommentera kod. De tyckte att verktyget inte tillförde något värde för att förstå koden och att kommentarerna var vagt formulerade och uppenbara istället för att vara informativa. Respondenten nämnde också att svårigheter med att förstå kod ofta berodde på kodstilen än på bristfälliga kommentarer.

Känslor kring ChatGPT och att automatisera kommentering av kod

Respondenten såg potentiella fördelar med att använda verktyg för att förbättra sammanhållningen bland mjukvaruutvecklings-team eller större grupper. Respondenten tyckte också att automatiserad kommentering av kod kunde vara användbart i vissa situationer för att etablera en mer standardiserad ansats. Dock betonade respondenten vikten av att verktyget var effektivt och löste problemet på ett tillfredsställande sätt för att vara verkligt värdefullt. Respondenten nämnde potentiella tidsbesparingar om ett verktyg kunde analysera och föreslå ändringar i återstående kod vid modifiering eller uppdatering av ett specifikt kodstycke. Respondenten visade öppenhet för verktyg som minskade tid och kostnader. De tog även upp möjligheten att integrera ChatGPT i programmeringsyrket, på liknande sätt som Google används för att söka efter information, men noterade behovet av ytterligare förbättringar och integration i olika miljöer.

5.3.3 Respondent C (JavaScript)

ChatGPT som ett verktyg

Enligt respondenten är majoriteten av kommentarerna överflödiga och saknar faktisk betydelse för förståelsen av koden. För att förbättra läsbarheten och förståelsen föreslår respondenten att en sista kommentar som beskriver hela funktionen skulle vara tillräcklig. Respondenten ser potentialen i att använda ChatGPT som ett verktyg för kommentering av kod i framtiden. Automatisering av kommentering ses som positivt, men respondenten betonar vikten av att ha kontroll över automatiseringsprocessen och inte förlita sig för mycket

på verktyget. Det betonas också att verktyg som ChatGPT kan vara till stor nytta genom att generera relevanta och informativa kommentarer för att förbättra enhetligheten i kommentarerna och spara tid. Dock måste man vara försiktig för att säkerställa att kommentarerna är korrekta och relevanta.

Hur presterar ChatGPT i nuläget

Enligt respondenten kan verktyg som ChatGPT vara till hjälp för att förbättra kvaliteten på koddokumentation och motverka mänskliga misstag. Dessa verktyg kan erbjuda mer informativa och fullständiga kommentarsförslag än vad enbart en människa kan producera. De kan också effektivisera uppdateringen av befintlig källkod och kommentarer. Genom standardisering kan verktygen bidra till att skapa objektiva och enhetliga förslag på kodändringar, vilket minskar risken för ofullständiga eller saknade kommentarer.

När det gäller ChatGPT:s prestationer i nuläget anser respondenten att majoriteten av dess kommentarer är överflödiga och inte tillför något värdefullt till förståelsen av koden. För att vara meningsfulla bör kommentarer vara relevanta och tillföra något av betydelse. Därför bör automatisering av kommentering användas sparsamt och fokuseras på att ge en helhetsbild av funktioner och klasser istället för att överflödigt kommentera varje rad kod.

Känslor kring ChatGPT och att automatisera kommentering av kod

Respondenten anser inte att människor kommer att blint förlita sig på verktyg som ChatGPT för att kommentera kod. Dock uttrycker respondenten en viss oro för att mindre erfarna programmerare kan komma att förlita sig för mycket på AI-modeller som ChatGPT och därigenom minska sin egen förmåga och kritiska tänkande. Det framhålls att det är viktigt att användare behåller sin kritiska förmåga och förstår att verktygen endast är hjälpmedel och inte ersättningar för mänsklig expertis och erfarenhet. Slutligen har respondenten inga specifika åsikter om huruvida Large Language Models (LLMs), som ChatGPT, kommer att vara nödvändiga för att vara konkurrenskraftiga i framtiden. Känslorna kring ChatGPT och automatiseringen av kommentering av kod är blandade. Respondenten är öppen för att använda verktyg som ChatGPT för att kommentera kod och ser fördelar med att automatisera processen för att spara tid och förbättra enhetligheten i kommentarerna. Det finns en tro på att ChatGPT kan bidra till att motverka mänskliga misstag genom att generera mer informativa och fullständiga kommentarer. Samtidigt betonar respondenten vikten av att behålla kritiskt tänkande och inte förlita sig för mycket på verktyg som ChatGPT för att undvika att minska kritiskt tänkande och självständiga problemlösningar.

5.3.4 Respondent D (Python)

ChatGPT som ett verktyg

Respondenten ser fördelar med att använda ChatGPT som ett verktyg för kommentering av kod. Respondenten anser att det kan vara användbart för att standardisera kommentarer och hitta potentiella felaktigheter. Hen ser också fördelar med att alla inom ett team eller projekt använder samma stil på kommentarer. Dock är respondenten försiktig med att förlita sig helt på AI för kommentering av kod, eftersom hen anser att mänsklig insikt och bedömning fortfarande behövs för att säkerställa korrekthet och meningsfullhet i kommentarerna.

Hur presterar ChatGPT i nuläget

Respondenten anser att ChatGPTs förmåga att kommentera kod varierar. Ibland ger det relevanta och användbara kommentarer, men i vissa fall ger det irrelevanta förslag. De påpekar att AI-modeller som ChatGPT har sina begränsningar och kan missa viktiga aspekter av kodens kontext. Respondenten betonar att vissa kommentarer kräver mänsklig insikt och förståelse för kontextet, vilket kan vara svårt att helt automatisera.

Känslor kring ChatGPT och att automatisera kommentering av kod

Respondenten ser fördelar med att använda verktyg vid kommentering av kod i framtiden, då det kan underlätta och påskynda processen. Automatisering av kommentering av kod kan vara fördelaktigt i vissa situationer, men vissa kommentarer kräver mänsklig insikt och förståelse för kontexten. Respondenten tycker att det är viktigt att ha en balans mellan automatisering och mänsklig kunskap för att säkerställa korrekta och meningsfulla kommentarer. Respondenten anser att ChatGPT kan vara ett användbart verktyg för att generera kommentarer och ge olika perspektiv på kod. Dock bör man enligt respondenten vara försiktig med att förlita sig helt på AI och se det som ett komplement till mänsklig expertis. Det är viktigt att fortsätta ha människors insikt och bedömning för att hantera komplexa problem och säkerställa att kommentarerna är korrekta och meningsfulla.

6. Diskussion och slutsatser

I det här avsnittet diskuteras studiens resultat med underlag av insamlad empiri samt det teoretiska ramverket. Det presenteras en övergripande bild och sammanfattning av de resultat som besvarar forskningsfrågorna.

6.1 Resultatdiskussion

Studiens syfte var att undersöka hur AI kan användas som verktyg för mjukvaruutvecklare. I studien avgränsas detta till kommentering av kod genom att använda ett verktyg som ChatGPT, en Large Language Model. Innebörden av att kommentera kod och huruvida AI, i synnerhet ChatGPT kan bistå mjukvaruutvecklare med att kommentera skriven kod genom ett LLM verktyg, har lett till uppkomst av andra implikationer. En av dessa är ifall det är fullt befogat att automatisera kommentering av kod. Respondenterna framförde vissa risker med att automatisera en sådan viktig aspekt som kommentering av kod inom mjukvaruutveckling. Det finns enligt majoriteten av respondenterna risk att mjukvaruutvecklare slutar tänka kritiskt och förlitar sig för mycket på olika AI-modeller.

Avhandlingens syfte gav upphov till nedanstående forskningsfråga som besvarats via det teoretiska ramverket och insamlad empiri.

- Hur kan användningen av Large Language Models underlätta för programmerare att kommentera källkod?

Resultaten från empirin påpekade en tydlig korrelation mellan popularitet i olika programmeringsspråk och hur väl ChatGPT kunde kommentera vald kod. Det är dock svårt att med endast samtliga observationer som underlag fastställa att ChatGPTs förmåga att kommentera kod handlar om popularitet för programmeringsspråket i fråga. Det skulle krävas fler studier och mer variation i framförallt variabler som används i instruktionen för att erhålla ett trovärdigt resultat. Exempelvis skulle liknande funktioner i olika språk framhäva en betydligt lättare jämförelse sinsemellan. Respondenterna blev uppmanade att välja ut källkod som var relativt komplex, men skribenterna gav inga direkta riktlinjer för detta, vilket innebär att komplexiteten kan variera markant mellan studiens respondenter.

Kommentering av kod är en stor del av mjukvaruutveckling, vilket både empiri och teori pekar mot. Majoriteten av studiens respondenter ansåg inte att ChatGPT i nuläget var lämplig att använda för att generera kommentarer. Det fanns flera brister som uppmärksammades av respondenterna, inte minst att många kommentarer ansågs vara överflödiga. Dessutom var kommentarerna för Erlang, enligt respondenten helt betydelselösa.

Studien visar att ChatGPT besitter förmågan att utföra både kodanalys och kommentering av kod. Trots detta finns det flera begränsningar i nuvarande verktyg. En av dessa begränsningar

är att användaren måste förhålla sig till en angiven teckenbegränsning vid inmatning samt tydligt specificera instruktioner för avsikten med koden. Som en följd av detta tas inte kodens fulla kontext i beaktning. Både Respondent A och Respondent B framhävde denna brist som betydande. Studien konstaterar att ChatGPT har potential som ett verktyg för kommentering av kod, trots vissa avsaknader i den nuvarande modellen och behovet av specifika instruktioner visar resultaten att modellen lyckas lägga till relativt passande kommentarer i de analyserade kodstyckena.

Framtagen teori och respondenterna i studien framhåller betydande brister i koddokumentation, vilket ofta resulterar i att källkod blir svårare att förstå för alla inblandade parter. De betonar även att kulturella skillnader inom olika projekt kan försvåra läsbarheten och förståelsen av koden genom kommentarer. Det finns inga tydliga indikationer från studien på att programmerarens roll kommer att genomgå en betydande förändring i framtiden. Trots brister i nuvarande modell, samt att vissa respondenter ansåg att kommentering av kod är en enkel och intuitiv process som inte behöver automatiseras, ansåg majoriteten av respondenterna att det är motiverat att standardisera kommentering av kod, men också att mänsklig granskning fortfarande behövs. Det är inte möjligt att enbart förlita sig på ett verktyg för att generera korrekta kommentarer, särskilt med tanke på den komplexitet som kan finnas i kontexten av omfattande kodstycken. Emellertid kan det vara lämpligt att standardisera uppdatering av kodkommentarer när källkod omstruktureras, eftersom studiens teori och empiri pekar på att det kan finnas subjektiva skillnader mellan utvecklare samt tidsbegränsningar och bristande motivation att uppdatera kommentarer inom olika teams och projekt.

I avslutande intervjun uttryckte respondenterna generellt att kommentarerna var överflödiga och saknade faktisk betydelse för förståelsen av koden. Vidare framfördes åsikter om potentialen i att använda ChatGPT i framtiden, men att det krävs en del förbättringar för att det ska kunna vara ett gediget tillvägagångssätt i arbetsrelaterade situationer. De potentiella nyttor som tillskrivs användandet av AI-modellen inom kommentering av kod är både enhetligheten och sparandet av tid. Vidare sågs potentialen i att automatiserad koddokumentation kan motverka mänskliga misstag.

6.2 Vidare forskning

Rapporten "Sparks of AGI" som genomfördes av Bubeck et al (2023) kan visa på små, gradvisa steg eller framsteg som forskare gör mot utvecklingen av AGI. Trots att AGI fortfarande är långt borta tror vissa forskare att dessa "sparks" eller genombrott inom områden som naturligt språkbearbetning, förstärkningsinlärning och självständig inlärning lägger grunden för framtida utveckling av AGI. Dock finns det fortfarande mycket forskning som behöver göras och många utmaningar som måste övervinnas innan riktig AGI kan förverkligas. Sammanfattningsvis visar denna studie att användning av ett verktyg, såsom ChatGPT, för att kommentera kodstycken är möjligt. Trots vissa brister i den nuvarande modellen har resultaten visat att modellen, tillsammans med väldefinierade instruktioner, kan

generera relativt passande kommentarer. Dessa slutsatser ger insikter och underlag för framtida forskning och utveckling inom området för kommentering av kod. Rapporten “Sparks of AGI” tillsammans med denna studie kan ge indikationer på att automatisering av koddokumentation kan komma att bli fullt möjligt, och kanske till och med fullt befogat i framtiden.

7. Källförteckning

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Yin Tat., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., Zhang, Y. (2023) Sparks of Artificial General Intelligence : Early experiments with GPT-4.
DOI: https://doi.org/10.48550/arXiv.2303.12712

Du, S. & Xie, C. (2021) Paradoxes of artificial intelligence in consumer markets: Ethical challenges and opportunities. *Journal of business research.*
DOI: https://doi.org/10.1016/j.jbusres.2020.08.024

Haleem, A., Javaid, M., Singh, R.P. (2023) An era of ChatGPT as a significant futuristic support tool: A study on features, abilities, and challenges. BenchCouncil Transactions on Benchmarks, Standards and Evaluations (2023), Article 100089
DOI: https://doi.org/10.1016/j.tbench.2023.100089

Huang, Y., Hu, X., Jia, N., Chen, X., Xiong,Y., Zheng,Z. (2019).
Learning Code Context Information to Predict Comment Locations.
DOI: 10.1109/TR.2019.2931725

Haenlein, M. & Kaplan, A. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. California Management Review, 61(4), 5-14.
DOI: https://doi.org/10.1177/0008125619864925

Jacobsen, D. I. & Andersson, S. (2017) Hur genomför man undersökningar? Introduktion till samhällsvetenskapliga metoder. 2 uppl. Lund: Studentlitteratur AB.

Kashefi, A. & Mukerji, T. (2023) ChatGPT for Programming Numerical Methods
DOI: https://doi.org/10.1615/JMachLearnModelComput.2023048492

Laramee. S, R. (2013). Bob´s Concise Coding Conventions.
https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=868154e591ee1c316c16b18cd6617915fae90b79

Makridakis, S. (2017) The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms. Futures Volume 90, June 2017, Pages 46-60
DOI: https://doi.org/10.1016/j.futures.2017.03.006 [2023-03-15]

Noy, C. (2008). Sampling Knowledge: The Hermeneutics of Snowball Sampling in Qualitative Research. International journal of social research methodology, 11(4), 327-344 .
DOI:https://doi.org/10.1080/13645570701401305.

Nigar M. Shafiq Surameery & Mohammed Y. Shakor (2023) Use chat GPT to solve programming bugs.
DOI: https://doi.org/10.55529/ijitc.31.17.22

OpenAI (2023) https://openai.com/product/gpt-4
[Hämtad: 2023-05-05]

Patel, R., & Davidson, B. (2011). Forskningsmetodikens grunder. Lund: Studentlitteratur AB.

Ross, S., Martinez, F., Houde, S., Muller, M., Weisz, J (2023). The Programmer's Assistans: Conversational Interaction with a Large Language Model for Software Development.
DOI : doi.org/10.48550/arXiv.2302.07080

Ray, P. P. (2023) ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems Volume 3, 2023, Pages 121-154.*
DOI: https://doi.org/10.1016/j.iotcps.2023.04.003

Smit, M., Gergel, B., Hoover, J., Stroulia, E. (2011). Code convention adherence in evolving software. DOI: 10.1109/ICSM.2011.6080819

Salviulo F, Scanniello G. (2014) Dealing with identifiers and comments in source code comprehension and maintenance:results from an ethnographically-informed study with students and professionals.
DOI:10.1145/2601248.2601251

Spatola, N & Normand, A. (2020). Human vs. machine: the psychological and behavioral consequences of being compared to an outperforming artificial agent.
DOI : 10.1007/s00426-020-01317-0

Spinelli, D (2010) Code Documentation
DOI: https://doi.org/10.1109/MS.2010.95

Skansholm, J. (2014) *Java direkt med swing*.
Upplaga 8:3., Studentlitteratur. ISBN 978-91-44-10431-7

Skärvad, P.-H., & Lundahl, U. (2016). *Utredningsmetodik*.
Lund: Studentlitteratur AB. ISBN 978-91-44-11457-6

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, L., Polosukhin, I. (2017). Attention Is All You Need
DOI: https://doi.org/10.48550/arXiv.1706.03762

Vailshery, L. S. (2022) Most used programming languages among developers worldwide as of 2022.
https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ [Hämtad: 2023-05-04]

Wen, F., Nagy, Csaba., Bavota,G., Lanza, M. (2019). A Large-Scale Empirical Study on Code-Comment Inconsistencies.
DOI: 10.1109/ICPC.2019.00019

Webbsidor

https://www.php.net/manual/en/index.php
[Hämtad: 2023-04-20]

https://wordpress.org/documentation/
[Hämtad: 2023-04-20]

https://www.erlang.org/doc/
[Hämtad: 2023-04-20]

https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ [Hämtad: 2023-05-04]

8. Bilagor

Intervjuguide Inledande Frågor

1. Vad har du för utbildning kopplat till programmeringsyrket?
2. Vilken tjänst har du idag?
3. Vilka är dina primära arbetsuppgifter inom programmering?
4. Har du någon expertis-område? (spetskompetens)
5. Vilket programmeringsspråk jobbar du med?
6. Vilken typ av utveckling jobbar du med? - t.ex back/front-end?
7. Hur många års erfarenhet besitter du?
8. Till vilken grad använder du kommentarer när du programmerar? Exempelvis, all kod, eller finns det funktioner som anses vara självbeskrivande?
9. Vid kommentering av kod, tar du hjälp av några hjälpmedel isåfall vad?
10. När du programmerar, kommenterar du koden enligt riktlinjer, eller mer ad-hoc?
    10.1 Vilka riktlinjer?
11. Om nej - Hur går du tillväga isf?
12. Hur viktigt tycker du det är att kommentera kod?
13. Har du varit med om tillfällen då kommentarer är bristfälliga, eller helt saknas? Berätta lite om det isf.
14. Vid uppdatering av kod, uppdaterar du även tillhörande kommentarer?
15. Hur tror du att Large language models kommer påverka ditt yrke i framtiden?

Intervjuguide Avslutande frågor

1. Hur väl kommenterade ChatGpt din kod?
2. Vilka tankar hade du kring att använda Chatgpt för att kommentera kod?
- underfråga för/nackdelar (tolka svaren som för/nackdel)
3. Kan du tänka dig att använda ett verktyg vid kommentering av kod i framtiden?
4. Vad tycker du om att automatisera kommentering av kod?
5. Tror du att ChatGPT kan motverka mänskliga misstag vad gäller kommentering av kod? Exempelvis att kommentarer är ofullständiga eller inte är skrivna alls?
6. Tror du att ChatGPT kan vara till hjälp för att automatisera uppdatering av kod (och på det sättet hjälpa till att undvika subjektiva olikheter bland utvecklare)
7. Tror du att det finns negativa faktorer att människor förlitar sig för mycket på olika verktyg som ChatGPT. (Exempelvis att kritiskt tänkande och självständiga problemlösningar minskar bland användare).

8. Tror du att LLMs kommer att bli en nödvändighet i framtiden för att vara konkurrenskraftig gentemot andra företag?
9. Tror du att din roll kommer att förändras inom olika aspekter baserat på nya teknologier som kan komma att förändra sättet man kommenterar kod på?
10. Tror du att ChatGPT kommer att vara en integrerad del i programmeringsyrket, likt google exempelvis där man söker upp information vid programmering?
11. Känns det som att ChatGPT har potential att hota ditt yrke som programmerare?

Bilaga Observationer

Respondent A (PHP)
Promt: Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: PHP for single-line comments and PHP for multi-line comments. Note that I only want comments within the code, and not below any display outputs. The code is divided into smaller chunks, when you are done with the first input, await the next and do the same operation as the last. Also await my input of code every time. Do not add any extra comments below the code display or above.

Respondent B (ERLANG)
Promt: Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: Erlang for single-line comments and Erlang for multi-line comments. Note that I only want comments within the code, and not below any display outputs. The code is divided into smaller chunks, when you are done with the first input, await the next and do the same operation as the last. Also await my input of code every time. Do not add any extra comments below the code display or above.

Respondent C (JavaScript)
Promt: Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: Javascript for single-line comments and Javascript for multi-line comments. Note that I only want comments within the code, and not below any display outputs. The code is divided into smaller chunks, when you are done with the first input, await the next and do the same operation as the last. Also await my input of code every time. Do not add any extra comments below the code display or above.

Respondent D (Python)
Hey ChatGPT, please help me add comments within my code for better readability and understanding, using the following comment convention: Python for single-line comments and Python for multi-line comments. Note that I only want comments within the code, and not below any display outputs. The code is divided into smaller chunks. When you are done

with the first input of code, await the next and do the same operation as the last. Also await my input of code every time.

JAVASCRIPT (FULL CODE, okommenterad)

```
(function() {
    let mode = 1;
    let mouseIsDown = false;
    let buttons = document.querySelectorAll('#obstacleMode, #startMode, #goalMode');
    let board = document.querySelector('#board');
    window.addEventListener('mousedown', function(e) {
        mouseIsDown = true;
        if (e.target.classList.contains('column')) {
            updateColumn(e.target);
        }
    });
    window.addEventListener('mouseup', function() { mouseIsDown = false; });

    function neutralizeColumn(columnElement) {
        let classes = ['obstacle', 'start', 'goal'];

        for (let i = 0; i < classes.length; i++) {
            columnElement.classList.remove(classes[i]);
        }
    }
    function updateObstacle(columnElement) {
        let isObstacle = columnElement.classList.contains('obstacle');
        neutralizeColumn(columnElement);
        if (!isObstacle) {
            columnElement.classList.add('obstacle');
```

```
    }
}

function updateStart(columnElement) {
    neutralizeColumn(columnElement);
    let starts = document.querySelectorAll('.start');
    for (let i = 0; i < starts.length; i++) {
        neutralizeColumn(starts[i]);
    }
    columnElement.classList.add('start');
}

function updateGoal(columnElement) {
    neutralizeColumn(columnElement);
    let starts = document.querySelectorAll('.goal');
    for (let i = 0; i < starts.length; i++) {
        neutralizeColumn(starts[i]);
    }
    columnElement.classList.add('goal');
}

function updateColumn(columnElement) {
    if (!mouseIsDown) {
        return;
    }

    switch (mode) {
```

```
        case 1:
            updateObstacle(columnElement)
            break;
        case 2:
            updateStart(columnElement)
            break;
        case 3:
            updateGoal(columnElement)
            break;
    }

    astar();
}

function renderColumn(colnr) {
    let e = document.createElement('div');
    e.classList.add('column');
    e.setAttribute('data-column', colnr);
    e.addEventListener('mouseenter', (e) => updateColumn(e.currentTarget));
    return e;
}

function renderRow(rownr) {
    let e = document.createElement('div');
    e.classList.add('row');
    e.setAttribute('data-row', rownr);
```

```
    for (let i = 0; i < columns; i++) {
        e.appendChild(renderColumn(i));
    }

    return e;
}

function getObstacles() {
    let obstacles = {};

    let oe = document.querySelectorAll('.obstacle');
    for (let i = 0; i < oe.length; i++) {
        let oex = parseInt(oe[i].getAttribute('data-column'));
        let oey = parseInt(oe[i].parentElement.getAttribute('data-row'));
        if (!isNaN(oex) && !isNaN(oey)) {
            obstacles[oey + 'x' + oex] = true;
        }
    }

    return obstacles;
}

function getNodeFromPos(row, column) {
    return document.querySelector(`.row[data-row="${row}"] .column[data-column="${column}"]`);
}

function getNeighbourNodes(pos) {
```

```
    return document.querySelectorAll(`
        .row[data-row="${pos.row-1}"] .column[data-column="${pos.column}"],

        .row[data-row="${pos.row}"] .column[data-column="${pos.column-1}"],
        .row[data-row="${pos.row}"] .column[data-column="${pos.column+1}"],

        .row[data-row="${pos.row+1}"] .column[data-column="${pos.column}"]
    `);
}

function getPos(column) {
    return {
        column: parseInt(column.getAttribute('data-column')),
        row: parseInt(column.parentElement.getAttribute('data-row'))
    }
}

function indexOfPos(src, row, column) {
    for (let i = 0; i < src.length; i++) {
        if (src[i].row === row && src[i].column === column) {
            return i;
        }
    }
    return -1;
}

function calcDistance(pos1, pos2) {
```

```
    return Math.abs(pos1.row - pos2.row) + Math.abs(pos1.column - pos2.column);
}

function lowestCostIndex(src) {
    if (src.length === 0) {
        return -1;
    }

    let lowest = 0;
    for (let i = 1; i < src.length; i++) {
        if (src[i].gScore + src[i].hScore < src[lowest].gScore + src[lowest].hScore) {
            lowest = i;
        }
    }
    return lowest;
}

function closeColumn(map, row, column, targetRow, targetColumn) {
    map[row + 'x' + column] = {
        targetRow: targetRow,
        targetColumn: targetColumn
    };
}

function getColumn(map, row, column) {
    return map[row + 'x' + column];
}
```

```
function clearPath() {
    let e = document.querySelectorAll('.path');
    for (let i = 0; i < e.length; i++) {
        e[i].classList.remove('path');
    }
}

function astarFinalize(closedset, aPos, bPos) {
    let path = [];
    let current = getColumn(closedset, bPos.row, bPos.column);

    while (current.targetRow !== aPos.row || current.targetColumn !== aPos.column) {
        if (!current) {
            return null;
        }

        path.push({
            row: current.targetRow,
            column: current.targetColumn
        });

        current = getColumn(closedset, current.targetRow, current.targetColumn);
    }
    path.reverse();
    return path;
}
```

```
function addCoordinate(openset, targetPos, fromPos, newPos) {
    let newPosIndexIndex = indexOfPos(openset, newPos.row, newPos.column);
    let data = {
        row: newPos.row,
        column: newPos.column,
        gScore: fromPos.gScore + 1,
        hScore: calcDistance(newPos, targetPos),
        targetRow: fromPos.row,
        targetColumn: fromPos.column
    };
    if (newPosIndexIndex === -1) {
        openset.push(data);
    } else if (openset[newPosIndexIndex].gScore + openset[newPosIndexIndex].hScore > data.gScore + data.hScore) {
        openset[newPosIndexIndex] = data;
    }
}

function astarSeek(aPos, bPos) {
    let openset = [{
        row: aPos.row,
        column: aPos.column,
        gScore: 0,
        hScore: calcDistance(aPos, bPos),
        targetRow: null,
        targetColumn: null
    }];
```

```
let closedset = {};

let obstacles = getObstacles();
while (true) {
    let lci = lowestCostIndex(openset);

    if (lci === -1) {
        console.log('error');
        return null;
    }

    let closest = openset.splice(lci, 1)[0];

    closeColumn(closedset, closest.row, closest.column, closest.targetRow, closest.targetColumn);

    if (closest.row == bPos.row && closest.column === bPos.column) {
        console.log('done');
        return astarFinalize(closedset, aPos, bPos);
    }

    let neighbours = getNeighbourNodes(closest);

    for (let i = 0; i < neighbours.length; i++) {
        let neighbour = getPos(neighbours[i]);
        if (getColumn(closedset, neighbour.row, neighbour.column)) {
            console.log('already closed');
```

```
            } else if (getColumn(obstacles, neighbour.row, neighbour.column)) {
                console.log('obstacle found');
            } else {
                console.log('possible coordinate found');
                addCoordinate(openset, bPos, closest, neighbour);
            }
        }
    }
}

function astar() {
    let pointA = document.querySelector('.start');
    let pointB = document.querySelector('.goal');
    if (pointA === null || pointB === null) {
        return;
    }

    let aPos = getPos(pointA);
    let bPos = getPos(pointB);

    let resp = astarSeek(aPos, bPos);

    clearPath();

    if (!resp) {
        return;
    }
```

```
    for (let i = 0; i < resp.length; i++) {
        getNodeFromPos(resp[i].row, resp[i].column).classList.add('path');
    }
}

function renderBoard() {
    board.innerHTML = '';
    if (rows <= 0 || columns <= 0) {
        return;
    }

    for (let i = 0; i < rows; i++) {
        board.appendChild(renderRow(i));
    }
}

function setActive(e) {
    for (let i = 0; i < buttons.length; i++) {
        buttons[i].classList.remove('active');
    }
    e.classList.add('active');
}

document.querySelector('#obstacleMode').addEventListener('click', function() {
    setActive(this);
    mode = 1;
```

```
    });

    document.querySelector('#startMode').addEventListener('click', function() {
        setActive(this);
        mode = 2;
    });

    document.querySelector('#goalMode').addEventListener('click', function() {
        setActive(this);
        mode = 3;
    });

    document.querySelector('#columns').addEventListener('input', function() {
        columns = parseInt(this.value) || 0;
        renderBoard();
    });

    document.querySelector('#rows').addEventListener('input', function() {
        rows = parseInt(this.value) || 0;
        renderBoard();
    });
})();
```

JAVASCRIPT (FULL CODE, kommenterad)

```
(function() {

   // Define variables
   let mode = 1; // Represents the current mode
   let mouseIsDown = false; // Indicates whether the mouse button is pressed or not
  // Get DOM elements
   let buttons = document.querySelectorAll('#obstacleMode, #startMode, #goalMode'); // Buttons for different modes
  let board = document.querySelector('#board'); // Board element
  // Event listener for mouse down
 window.addEventListener('mousedown', function(e) {
   mouseIsDown = true; // Set mouseIsDown to true when mouse button is pressed
   if (e.target.classList.contains('column')) {
    updateColumn(e.target); // Call a function to update the column
     }
   });
   // Event listener for mouse up
   window.addEventListener('mouseup', function() {
      mouseIsDown = false; // Set mouseIsDown to false when mouse button is released
 });
  // Function to neutralize a column
 function neutralizeColumn(columnElement) {
  let classes = ['obstacle', 'start', 'goal']; // Classes to remove from the column element
           for (let i = 0; i < classes.length; i++) {
 columnElement.classList.remove(classes[i]); // Remove each class from the column element
   }
```

```
    }
})();
function updateObstacle(columnElement) {
    let isObstacle = columnElement.classList.contains('obstacle');
    neutralizeColumn(columnElement); // Reset the column element
    if (!isObstacle) {
        columnElement.classList.add('obstacle'); // Add the 'obstacle' class to the column element
    }
}
function updateStart(columnElement) {
    neutralizeColumn(columnElement); // Reset the column element
    let starts = document.querySelectorAll('.start'); // Get all elements with the 'start' class
    for (let i = 0; i < starts.length; i++) {
        neutralizeColumn(starts[i]); // Reset each 'start' element
    }
```

```
    columnElement.classList.add('start'); // Add the 'start' class to the column element
}
function updateGoal(columnElement) {
    neutralizeColumn(columnElement); // Reset the column element
    let goals = document.querySelectorAll('.goal'); // Get all elements with the 'goal' class
    for (let i = 0; i < goals.length; i++) {
        neutralizeColumn(goals[i]); // Reset each 'goal' element
    }
    columnElement.classList.add('goal'); // Add the 'goal' class to the column element
}
function updateColumn(columnElement) {
    if (!mouseIsDown) {
        return; // Exit the function if the mouse button is not pressed
    }
    switch (mode) {
        case 1:
```

```
            updateObstacle(columnElement); // Call the function to update as an obstacle
            break;
        case 2:
            updateStart(columnElement); // Call the function to update as a start point
            break;
        case 3:
            updateGoal(columnElement); // Call the function to update as a goal point
            break;
    }
    astar(); // Call the 'astar' function
}
function renderColumn(colnr) {
    let e = document.createElement('div');
    e.classList.add('column'); // Add the 'column' class to the element
    e.setAttribute('data-column', colnr); // Set the 'data-column' attribute
    e.addEventListener('mouseenter', (e) => updateColumn(e.currentTarget)); // Add event listener for mouse enter
```

```
    return e; // Return the created element
}
function renderRow(rownr) {
    let e = document.createElement('div');
    e.classList.add('row'); // Add the 'row' class to the element
    e.setAttribute('data-row', rownr); // Set the 'data-row' attribute
    for (let i = 0; i < columns; i++) {
        e.appendChild(renderColumn(i)); // Append a rendered column element to the row element
    }
    return e; // Return the created element
}
function getObstacles() {
    let obstacles = {};
    let obstacleElements = document.querySelectorAll('.obstacle'); // Get all elements with the 'obstacle' class
    for (let i = 0; i < obstacleElements.length; i++) {
        let obstacleX = parseInt(obstacleElements[i].getAttribute('data-column')); // Get the 'data-column' attribute value as X coordinate
```

```
        let obstacleY = parseInt(obstacleElements[i].parentElement.getAttribute('data-row')); // Get the parent element's 'data-row' attribute value as Y coordinate
        if (!isNaN(obstacleX) && !isNaN(obstacleY)) {
            obstacles[obstacleY + 'x' + obstacleX] = true; // Add the obstacle coordinates to the 'obstacles' object
        }
    }
    return obstacles; // Return the 'obstacles' object
}
function getNodeFromPos(row, column) {
    return document.querySelector(`.row[data-row="${row}"] .column[data-column="${column}"]`);
}
function getNeighbourNodes(pos) {
    return document.querySelectorAll(`
        .row[data-row="${pos.row-1}"] .column[data-column="${pos.column}"],
        .row[data-row="${pos.row}"] .column[data-column="${pos.column-1}"],
        .row[data-row="${pos.row}"] .column[data-column="${pos.column+1}"],
        .row[data-row="${pos.row+1}"] .column[data-column="${pos.column}"]
```

```
    `);
}


function getPos(column) {
    return {
        column: parseInt(column.getAttribute('data-column')), // Get the 'data-column' attribute value as column
        row: parseInt(column.parentElement.getAttribute('data-row')) // Get the parent element's 'data-row' attribute value as row
    };
}
function indexOfPos(src, row, column) {
    for (let i = 0; i < src.length; i++) {
        if (src[i].row === row && src[i].column === column) {
            return i; // Return the index of the position if found in the source array
        }
    }
    return -1; // Return -1 if the position is not found in the source array
```

```
}

function calcDistance(pos1, pos2) {
    return Math.abs(pos1.row - pos2.row) + Math.abs(pos1.column - pos2.column);
}

function lowestCostIndex(src) {
    if (src.length === 0) {
        return -1; // Return -1 if the source array is empty
    }
    let lowest = 0; // Set the initial lowest index to 0
    for (let i = 1; i < src.length; i++) {
        if (src[i].gScore + src[i].hScore < src[lowest].gScore + src[lowest].hScore) {
            lowest = i; // Update the lowest index if the current element has a lower combined score (gScore + hScore)
        }
    }
    return lowest; // Return the index of the lowest-cost node
}
```

```
function closeColumn(map, row, column, targetRow, targetColumn) {
    map[row + 'x' + column] = {
        targetRow: targetRow,
        targetColumn: targetColumn
    };
}
function getColumn(map, row, column) {
    return map[row + 'x' + column];
}
function clearPath() {
    let pathElements = document.querySelectorAll('.path'); // Get all elements with the 'path' class
    for (let i = 0; i < pathElements.length; i++) {
        pathElements[i].classList.remove('path'); // Remove the 'path' class from each element
    }
}
```

```
function astarFinalize(closedset, aPos, bPos) {
    let path = [];
    let current = getColumn(closedset, bPos.row, bPos.column);
    while (current.targetRow !== aPos.row || current.targetColumn !== aPos.column) {
        if (!current) {
            return null; // Return null if the current element is not found
        }
        path.push({
            row: current.targetRow,
            column: current.targetColumn
        }); // Add the target position to the path

        current = getColumn(closedset, current.targetRow, current.targetColumn); // Update the current element with the next position
    }
    path.reverse(); // Reverse the path array to get the correct order
    return path; // Return the finalized path
```

```
}

function addCoordinate(openset, targetPos, fromPos, newPos) {
    let newPosIndex = indexOfPos(openset, newPos.row, newPos.column);
    let data = {
        row: newPos.row,
        column: newPos.column,
        gScore: fromPos.gScore + 1,
        hScore: calcDistance(newPos, targetPos),
        targetRow: fromPos.row,
        targetColumn: fromPos.column
    };
    if (newPosIndex === -1) {
        openset.push(data); // Add the new coordinate to the openset if it doesn't exist
    } else if (openset[newPosIndex].gScore + openset[newPosIndex].hScore > data.gScore + data.hScore) {
        openset[newPosIndex] = data; // Update the coordinate in the openset if the new path has a lower combined score (gScore + hScore)
    }
```

```
}

function astarSeek(aPos, bPos) {

    let openset = [{

        row: aPos.row,

        column: aPos.column,

        gScore: 0,

        hScore: calcDistance(aPos, bPos),

        targetRow: null,

        targetColumn: null

    }];

}

// Initialize the closed set as an empty object

let closedset = {};

// Get the obstacles from the board

let obstacles = getObstacles();

// Perform the A* search loop
```

```
while (true) {
    // Find the index of the node with the lowest cost in the open set
    let lci = lowestCostIndex(openset);
    if (lci === -1) {
        // If there are no more nodes in the open set, return null (error)
        console.log('error');
        return null;
    }
    // Get the node with the lowest cost from the open set
    let closest = openset.splice(lci, 1)[0];
    // Add the current node to the closed set
    closeColumn(closedset, closest.row, closest.column, closest.targetRow, closest.targetColumn);
    // Check if the current node is the goal node
    if (closest.row == bPos.row && closest.column === bPos.column) {
        // If the goal is reached, return the finalized path
        console.log('done');
```

```
    return astarFinalize(closedset, aPos, bPos);
}
// Get the neighboring nodes of the current node
let neighbours = getNeighbourNodes(closest);
// Iterate through the neighboring nodes
for (let i = 0; i < neighbours.length; i++) {
    let neighbour = getPos(neighbours[i]);
    if (getColumn(closedset, neighbour.row, neighbour.column)) {
        // If the neighboring node is already in the closed set, log "already closed"
        console.log('already closed');
    } else if (getColumn(obstacles, neighbour.row, neighbour.column)) {
        // If the neighboring node is an obstacle, log "obstacle found"
        console.log('obstacle found');
    } else {
        // If the neighboring node is a possible coordinate, log "possible coordinate found" and add it to the open set
        console.log('possible coordinate found');
```

```
            addCoordinate(openset, bPos, closest, neighbour);
        }
    }
}
// A* algorithm implementation
function astar() {
    // Get the start and goal points from the board
    let pointA = document.querySelector('.start');
    let pointB = document.querySelector('.goal');
    // Check if both start and goal points exist
    if (pointA === null || pointB === null) {
        return;
    }
    // Get the positions of the start and goal points
    let aPos = getPos(pointA);
    let bPos = getPos(pointB);
```

```
    // Perform A* search to find the optimal path
    let resp = astarSeek(aPos, bPos);
    // Clear any previously rendered path
    clearPath();
    // If no path is found, return
    if (!resp) {
        return;
    }
    // Render the path on the board
    for (let i = 0; i < resp.length; i++) {
        getNodeFromPos(resp[i].row, resp[i].column).classList.add('path');
    }
}
// Render the game board
function renderBoard() {
    // Clear the board
```

```
    board.innerHTML = '';
    // Check if the number of rows and columns is valid
    if (rows <= 0 || columns <= 0) {
        return;
    }
    // Render each row and column on the board
    for (let i = 0; i < rows; i++) {
        board.appendChild(renderRow(i));
    }
}
// Set the active mode and update the mode variable
function setActive(e) {
    for (let i = 0; i < buttons.length; i++) {
        buttons[i].classList.remove('active');
    }
    e.classList.add('active');
```

```
}

// Event listener for obstacle mode button

document.querySelector('#obstacleMode').addEventListener('click', function() {

    setActive(this);

    mode = 1;

});

function astar() {

    // Get the start and goal points

    let pointA = document.querySelector('.start');

    let pointB = document.querySelector('.goal');

    if (pointA === null || pointB === null) {

        return;

    }

    let aPos = getPos(pointA);

    let bPos = getPos(pointB);

    // Perform A* search
```

```
    let resp = astarSeek(aPos, bPos);
    // Clear previous path
    clearPath();
    if (!resp) {
        return;
    }
    // Add 'path' class to the nodes in the path
    for (let i = 0; i < resp.length; i++) {
        getNodeFromPos(resp[i].row, resp[i].column).classList.add('path');
    }
}
function renderBoard() {
    // Clear the board
    board.innerHTML = '';
    if (rows <= 0 || columns <= 0) {
        return;
```

```
    }

    // Render rows and columns

    for (let i = 0; i < rows; i++) {

        board.appendChild(renderRow(i));

    }
}

function setActive(e) {

    // Remove 'active' class from all buttons and add it to the clicked button

    for (let i = 0; i < buttons.length; i++) {

        buttons[i].classList.remove('active');

    }

    e.classList.add('active');

}

// Event listener for obstacle mode button
document.querySelector('#obstacleMode').addEventListener('click', function() {

    setActive(this);

    mode = 1;
```

```
});

document.querySelector('#startMode').addEventListener('click', function() {
    // Set the active mode to start mode
    setActive(this);
    mode = 2;
});

document.querySelector('#goalMode').addEventListener('click', function() {
    // Set the active mode to goal mode
    setActive(this);
    mode = 3;
});

document.querySelector('#columns').addEventListener('input', function() {
    // Update the number of columns and render the board
    columns = parseInt(this.value) || 0;
    renderBoard();
});
```

```
document.querySelector('#rows').addEventListener('input', function() {
    // Update the number of rows and render the board
    rows = parseInt(this.value) || 0;
    renderBoard();
});
})();
```

Erlang (FULL Code, okommenterad)

```
setopts_active(#state{socket=Socket, transport=Transport, opts=Opts}) ->
        N = maps:get(active_n, Opts, 100),
        Transport:setopts(Socket, [{active, N}]).
loop(State=#state{parent=Parent, socket=Socket, transport=Transport,
        opts=Opts, timer=TimerRef, children=Children}, Buffer) ->
        Messages = Transport:messages(),
        InactivityTimeout = maps:get(inactivity_timeout, Opts, 300000),
        receive
        {OK, Socket, Data} when OK =:= element(1, Messages) ->
        parse(set_idle_timeout(State), << Buffer/binary, Data/binary >>);
        {Closed, Socket} when Closed =:= element(2, Messages) ->
        Reason = case State#state.http2_status of
                closing -> {stop, closed, 'The client is going away.'};
                _ -> {socket_error, closed, 'The socket has been closed.'}
        end,
        terminate(State, Reason);
        {Error, Socket, Reason} when Error =:= element(3, Messages) ->
        terminate(State, {socket_error, Reason, 'An error has occurred on the socket.'});
        {Passive, Socket} when Passive =:= element(4, Messages);
        Passive =:= tcp_passive; Passive =:= ssl_passive ->
        setopts_active(State),
        loop(State, Buffer);
```

```
{'EXIT', Parent, shutdown} ->
Reason = {stop, {exit, shutdown}, 'Parent process requested shutdown.'},
loop(initiate_closing(State, Reason), Buffer);
{'EXIT', Parent, Reason} ->
terminate(State, {stop, {exit, Reason}, 'Parent process terminated.'});
{system, From, Request} ->
sys:handle_system_msg(Request, From, Parent, ?MODULE, [], {State, Buffer});
{timeout, TimerRef, idle_timeout} ->
terminate(State, {stop, timeout,
        'Connection idle longer than configuration allows.'});
{timeout, Ref, {shutdown, Pid}} ->
cowboy_children:shutdown_timeout(Children, Ref, Pid),
loop(State, Buffer);
end.
```

Erlang (FULL code, kommenterad)

```
%% Function to set active options on the socket
setopts_active(#state{socket=Socket, transport=Transport, opts=Opts}) ->
    N = maps:get(active_n, Opts, 100),
    Transport:setopts(Socket, [{active, N}]).
%% Main loop function
loop(State=#state{parent=Parent, socket=Socket, transport=Transport,
    opts=Opts, timer=TimerRef, children=Children}, Buffer) ->
    Messages = Transport:messages(),
    InactivityTimeout = maps:get(inactivity_timeout, Opts, 300000),
    receive
    %% Handling incoming data
    {OK, Socket, Data} when OK =:= element(1, Messages) ->
    parse(set_idle_timeout(State), << Buffer/binary, Data/binary >>);
    %% Handling socket closure
    {Closed, Socket} when Closed =:= element(2, Messages) ->
    Reason = case State#state.http2_status of
        closing -> {stop, closed, 'The client is going away.'};
        _ -> {socket_error, closed, 'The socket has been closed.'}
    end,
    terminate(State, Reason);
    %% Handling socket errors
    {Error, Socket, Reason} when Error =:= element(3, Messages) ->
```

```
terminate(State, {socket_error, Reason, 'An error has occurred on the socket.'});
%% Setting passive options on the socket
{Passive, Socket} when Passive =:= element(4, Messages);
Passive =:= tcp_passive; Passive =:= ssl_passive ->
setopts_active(State),
loop(State, Buffer);
%% Handling shutdown request from parent process
{'EXIT', Parent, shutdown} ->
Reason = {stop, {exit, shutdown}, 'Parent process requested shutdown.'},
loop(initiate_closing(State, Reason), Buffer);
%% Handling termination of parent process
{'EXIT', Parent, Reason} ->
terminate(State, {stop, {exit, Reason}, 'Parent process terminated.'});
%% Handling system messages
{system, From, Request} ->
sys:handle_system_msg(Request, From, Parent, ?MODULE, [], {State, Buffer});
%% Handling idle timeout
{timeout, TimerRef, idle_timeout} ->
terminate(State, {stop, timeout,
        'Connection idle longer than configuration allows.'});
%% Handling shutdown timeout
{timeout, Ref, {shutdown, Pid}} ->
cowboy_children:shutdown_timeout(Children, Ref, Pid),
loop(State, Buffer);
end.
```

Högskolan i Borås är en modern högskola mitt i city. Vi bedriver utbildningar inom ekonomi och informatik, biblioteks- och informationsvetenskap, mode och textil, beteendevetenskap och lärarutbildning, teknik samt vårdvetenskap.

På institutionen Handels- och IT-högskolan (HIT) har vi tagit fasta på studenternas framtida behov. Därför har vi skapat utbildningar där anställningsbarhet är ett nyckelord. Ämnesintegration, helhet och sammanhang är andra viktiga begrepp. På institutionen råder en närhet, såväl mellan studenter och lärare som mellan företag och utbildning.

Våra **ekonomiutbildningar** ger studenterna möjlighet att lära sig mer om olika företag och förvaltningar och hur styrning och organisering av dessa verksamheter sker. De får även lära sig om samhällsutveckling och om organisationers anpassning till omvärlden. De får möjlighet att förbättra sin förmåga att analysera, utveckla och styra verksamheter, oavsett om de vill ägna sig åt revision, administration eller marknadsföring. Bland våra **IT-utbildningar** finns alltid något för dem som vill designa framtidens IT-baserade kommunikationslösningar, som vill analysera behov av och krav på organisationers information för att designa deras innehållsstrukturer, bedriva integrerad IT- och affärsutveckling, utveckla sin förmåga att analysera och designa verksamheter eller inrikta sig mot programmering och utveckling för god IT-användning i företag och organisationer.

Forskningsverksamheten vid institutionen är såväl professions- som design- och utvecklingsinriktad. Den övergripande forskningsprofilen för institutionen är handels- och tjänsteutveckling i vilken kunskaper och kompetenser inom såväl informatik som företagsekonomi utgör viktiga grundstenar. Forskningen är välrenommerad och fokuserar på inriktningarna affärsdesign och Co-design. Forskningen är också professionsorienterad, vilket bland annat tar sig uttryck i att forskningen i många fall bedrivs på aktionsforskningsbaserade grunder med företag och offentliga organisationer på lokal, nationell och internationell arena. Forskningens design och professionsinriktning manifesteras också i InnovationLab, som är institutionens och Högskolans enhet för forskningsstödjande systemutveckling.

HÖGSKOLAN I BORÅS
VETENSKAP FÖR PROFESSION

Besöksadress: Järnvägsgatan 5 · Postadress: Allégatan 1, 501 90 Borås
Tfn: 033-435 40 00 · E-post: inst.hit@hb.se · Webb: www.hb.se/hit