

Elias Eskelinen

GENERATIIVISEN TEKOÄLYN HYÖDYNTÄMINEN OHJELMISTOKEHITYKSESSÄ

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Joulukuu 2025

TIIVISTELMÄ

Elias Eskelinen: Generatiivisen tekoälyn hyödyntäminen ohjelmistokehityksessä
Kandidaatintutkielma
Tampereen yliopisto
Informaatioteknologian ja viestinnän tiedekunta
Marraskuu 2025

Tämä tutkielma on kirjallisuuskatsaus, jonka tarkoituksena on selvittää, miten generatiivinen tekoäly vaikuttaa ohjelmistokehitysprosessiin ja sen eri vaiheisiin sekä millaisia mahdollisuuksia ja haasteita tekoälytyökalut tuovat mukanaan. Työ perustuu viime vuosina (2023–2025) julkaistuihin kirjallisuuteen aiheesta ja keskittyy erityisesti suurten kielimallien rooliin ohjelmistokehitysprosessissa. Tulokset osoittavat, että generatiivista tekoälyä voidaan käyttää apuna laajalti erilaisissa ohjelmistokehitykseen liittyvissä tehtävissä. Generatiivinen tekoäly voi nopeuttaa ja parantaa ohjelmistokehityksen laatua tukemalla vaatimusten määrittelyä, suunnittelua, ohjelmointia ja testausta eri tavoin. Vaatimusten määrittelyssä tekoäly voi analysoida vaatimuksia, perustella jos kriteerit eivät täyty ja ehdottaa parannuksia tai generoida vaatimusmäärittelydokumentin. Suunnittelussa tekoäly keventää käyttäjän kognitiivista kuormitusta ja edistää suunnitteluprosessin luovuutta sekä työn laatua. Toteutuksessa tekoälyä käyttävät ohjelmoijat vähentävät merkittävästi työaika koodin kirjoittamisessa, dokumentoinnissa ja refaktoroinnissa, suoriutuvat todennäköisemmin monimutkaisista tehtävistä tavoiteajassa sekä kokevat parempaa työtyytyväisyyttä ja flow-tilaa. Testauksessa tekoäly voi tuottaa testitapauksia kehoitteiden ja käyttötapauskaavioiden avulla eri sovellustasoille, nopeuttaen testien luontia ja parantaen kattavuutta. Tekoälytyökalun käyttö kuitenkin edellyttää ohjelmoijalta osaamista ja tuntemusta työkalusta, esimerkiksi kehoitteen muotoilussa ja tekoälyn tuottamien tulosten kriittisessä arvioinnissa. Mahdollisia generatiivisen tekoälyn käyttämiseen liittyviä ongelmia ovat tietoturva, immateriaalioikeudet, vinoutunut data ja tekoälyn tuottaman koodin luotettavuus. Työ osoittaa, että generatiivinen tekoäly tarjoaa huomattavaa potentiaalia ohjelmistokehityksen tehostamiseen, mutta sen vastuullinen ja tehokas hyödyntäminen vaatii harkittua käyttöönottoa ja osaamista käyttäjältä.

Avainsanat: Generatiivinen tekoäly, ohjelmistokehitys, ohjelmistoprosessi, kielimallit

Tämän julkaisun alkuperäisyys on tarkistettu Turnitin Originality Check -ohjelmalla.

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

- Ei
- Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni tutkielmanprosessin aikana seuraavia tekoälysovelluksia:

Tekoälysovellusten nimet ja versiot: ScopusAI, ChatGPT GPT-5

Käyttötarkoitus: ScopusAI:ta on hyödynnetty lähteiden etsinnässä ja koko työn rakenteen ideoinnissa. GPT-5:ta on käytetty kieliasun tarkistukseen.

Osiot, joissa tekoälyä on käytetty: ScopusAI lähteiden etsintä ja suunnittelu, kattaa koko työn. GPT-5 kieliasun tarkistusta on tehty paikoitellen koko tekstin osalta.

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

SISÄLLYSLUETTELO

| | |
|---|----|
| 1. JOHDANTO | 1 |
| 2. MENETELMÄT | 2 |
| 2.1 Keskeiset käsitteet ja konseptit | 2 |
| 2.2 Aineistonhankinta | 2 |
| 3. OHJELMISTOKEHITYS | 4 |
| 3.1 Ohjelmiston vaatimukset | 4 |
| 3.2 Ohjelmiston suunnittelu | 5 |
| 3.3 Ohjelmiston toteutus | 5 |
| 3.4 Ohjelmiston testaus | 6 |
| 4. GENERATIIVISEN TEKOÄLYN ROOLI OHJELMISTOKEHITYKSESSÄ | 7 |
| 4.1 Tekoäly ja vaatimukset | 8 |
| 4.2 Tekoäly ja suunnittelu | 8 |
| 4.3 Tekoäly ja toteutus | 9 |
| 4.4 Tekoäly ja testaus | 11 |
| 4.5 Yhteenveto | 12 |
| 5. GENERATIIVISEN TEKOÄLYN HAASTEET | 14 |
| 6. YHTEENVETO | 15 |
| LÄHTEET | 16 |

1. JOHDANTO

Työn tavoitteena on tutkia, miten kehittyvä generatiivinen tekoäly vaikuttaa ohjelmistokehitykseen ja sen osa-alueisiin sekä millaisia tekijöitä tekoälyn hyödyntämisessä tulee ottaa huomioon. Tarkastelun kohteena ovat ohjelmistokehitysprosessin eri vaiheet sekä generatiivisen tekoälyn tarjoamat mahdollisuudet tehostaa näitä vaihteita, niin ohjelmoijan työn kuin laajemman prosessin tasolla. Työssä keskitytään siihen, mitä kirjallisuus kertoo tekoälytyökalujen mahdollisuuksista tehostaa ohjelmistokehitystä sen eri vaiheissa. Työn tutkimuskysymys on: Kuinka generatiivista tekoälyä voidaan hyödyntää ohjelmistokehityksen eri vaiheissa?

Generatiivisen tekoälyn hyödyntäminen on nykyisessä laajuudessaan uutta ja teknologian mahdollisuudet sekä käyttötavat ovat jatkuvassa kehityksessä. Suuret kielimallit ovat saavuttaneet räjähdysmäisen suosion [1]. Tämä takaa niiden vaikutuksen tulevaisuuden työnäkymiin tavoilla, jotka eivät vielä ole täysin ennakoitavissa. Aiheesta tehty tutkimus on nopeasti kehittyvää sekä erityisen kiinnostavaa mahdollisena tulevaisuuden ohjelmistokehityksen alan suunnan ennustajana.

Oikein käytettynä tekoälytyökalut ovat osoittautuneet paljon ohjelmoijan työntekoa tehostaviksi [1]. Suurempia ohjelmistoprojekteja toteuttaessa on kuitenkin otettava huomioon paljon suurempi kirjo muuttujia kuin yhden ohjelmoijan henkilökohtainen ohjelmointityö. On siis ensiarvoisen tärkeää selvittää, miten suuremman ohjelmiston kehityksen eri vaiheissa on mahdollista hyödyntää tekoälytyökaluja ja mitä vaatimuksia tämä asettaa sekä ohjelmoijille että kehitettävälle ohjelmistolle.

Tämän työn tarkoituksena on tehdä katsaus aiheesta tehtyyn tutkimukseen ja selvittää tämän avulla, kuinka ohjelmistojen ja ohjelmoijien tulee vastata tekoälytyökalujen asettamiin vaatimuksiin sekä haasteisiin niiden kehityksen edetessä ja käytön lisääntyessä.

Työn rakenne on seuraavanlainen. Ensimmäinen luku toimii johdantona aiheeseen. Toisessa luvussa käydään tutkielman menetelmiä, käsitteitä ja konsepteja sekä aineistonhankintaprosessia. Kolmannessa luvussa kuvataan ohjelmistokehitysprosessia ja sen vaihteita, järjestyksessä omissa alaluvuissaan. Neljäs luku tarkastelee tekoälyn käsitettä ja kokoaa tutkimustietoa siitä, miten tekoälyä voidaan hyödyntää ohjelmistokehityksen eri vaiheissa, käyden vaiheet järjestyksessä läpi omissa alaluvuissaan. Viidennessä luvussa käsitellään tekoälyn käytön haasteita. Kuudes luku tarjoaa yhteenvedon ja keskustelua tutkimuksen keskeisistä teemoista.

2. MENETELMÄT

2.1 Keskeiset käsitteet ja konseptit

Työ toteutetaan kirjallisuuskatsauksena ja keskittyy generatiivisen tekoälyn tarkasteluun ohjelmistoprosessin kontekstissa, erityisesti suurten kielimallien hyödyntämiseen työn tehostamisessa. Generatiivinen tekoäly (engl. GenAI, Generative Artificial Intelligence) viittaa neuroverkoilla toimivaan malliin, jolla voidaan tunnistaa säännönmukaisuuksia alkuperäisdatassa ja näiden pohjalta luoda uutta sisältöä [2].

Ohjelmistokehitys on murroksessa suurten kielimallien (engl. LLM, Large Language Models) yleistymisen myötä. Suurilla kielimalleilla tarkoitetaan tekoälymalleja, jonka koulutusdataa on paljon ja ne pystyvät generoimaan tietoa luonnollisella kielellä [3]. Ohjelmoijilla on jo käytössä kielimalleja avustajina, tulkkeina ja erinäisinä automatisoituina GPT-malleina optimoimassa kehitystyötä sekä työtehtäviä [1]. GPT-mallilla (Generative Pre-trained Transformer) tarkoitetaan suurta, Transformer-arkkitehtuuriin perustuvaa kielimallia, joka on esikoulutettu tuottamaan ja ymmärtämään luonnollista kieltä.

Työ pyrkii selvittämään, miten generatiivista tekoälyä on käytetty ja mahdollisesti voitaisiin tulevaisuudessa käyttää tehostamaan ohjelmistokehityksen osa-alueita. On myös tärkeää huomioida millaisia uusia vaatimuksia generatiivisen tekoälyn käyttö asettaa ohjelmistokehityksen menetelmille. Generatiivisen tekoälyn hyödyntäminen esimerkiksi ketterän kehityksen Scrum-prosessin tukena voi avustaa muun muassa käyttäjätarinoiden luonnissa, tehtävien vaativuuden arvioinnissa, tutkimusapulaisena sekä etenemisen seurannassa [4].

Toinen tarkasteltava puoli on ohjelmoijan työnkuva, kuinka se kehittyy ja millaisia uusia vaatimuksia generatiivisen tekoälyn hyödyntäminen ohjelmistoprosessissa asettaa ohjelmoijalle. On jo viitteitä siitä, että tekoälytyökaluja hyödyntävät ohjelmoijat käyttävät enemmän aikaa koodin tarkastamiseen kuin itse kirjoittamiseen [5]. Tällaiset muutokset ohjelmoijan työnkuvaan mahdollisesti lisääntyvät tekoälytyökalujen kehittyessä ja niiden käytön yleistyessä.

2.2 Aineistonhankinta

Aineistonhankinnassa käytettyjä tietokantoja olivat Springer Nature link, Academic Search Ultimate (EBSCO), Computer Science Database (ProQuest), IEEE Xplore - IEEE

Electronic Library (IEL), ACM Digital Library, ResearchGate sekä Tampereen yliopiston Andor. Aineistonhankinnassa on käytetty lisäksi ScopusAI tekoälytyökalua. Tietokantoihin tehdyt haut ovat muodostettu seuraavista avainsanoista: ”Generative AI”, ”Software”, ”Phase”, ”Requirements”, ”Design”, ”Development”, ”Testing” ja ”Agile”. Andor-palvelusta on haettu pääsiallisesti maksumuurin takana olevaa kirjallisuutta.

Generatiivisesta tekoälystä ja kielimalleista löydetty tutkimukset rajautuivat vuosille 2023–2025 ja ne ovat konferenssijulkaisuihin painottuvia, johtuen tutkimusaiheen nopeasta kehityskulusta. Vertaisarvioituja tieteellisiä artikkeleita on kuitenkin pyritty priorisoimaan muiden julkaisujen edelle, kun mahdollista. Tutkimukset valittiin sen perusteella, että ne käsittelivät generatiivisen tekoälyn vaikutusta ohjelmistokehitykseen tai johonkin sen osa-alueeseen. Toinen kriteeri oli, että tutkimuksissa oli saatu esille tuloksia, kuinka generatiivisen tekoälyn käyttö vaikutti ohjelmistokehitykseen. Poissuljettuja tutkimuksia olivat ne, joiden painopiste oli yksinomaan tekoälyavusteisessa ohjelmoinnissa, eri tekoälytyökalujen tehokkuuden vertailussa.

Teokset luettiin useassa eri vaiheessa, aloittaen tiivistelmän ja lopputulosten perusteella kartoittamalla yleiset teemat, minkä jälkeen tutkimusta tarkasteltiin perusteellisemmin. Tarkastelu keskittyi tekoälytyökalujen vaikutukseen ohjelmistokehityksen kulkuun, käyttäjän ja mallin väliseen vuorovaikutukseen sekä esiintyneisiin haasteisiin tekoälytyökalujen käytössä. Teemojen avulla tutkimusten havaintoja tarkasteltiin suhteessa toisiin, jotta voitiin tunnistaa toistuvia ilmiöitä sekä mahdollisia jatkotutkimuksen kohteita.

3. OHJELMISTOKEHITYS

Ohjelmistokehitys on järjestelmällinen prosessi toisiinsa liittyviä toimintoja, jotka johtavat ohjelmistojärjestelmän tuotantoon [6]. Tähän prosessiin sisältyy jossakin muodossa käyttäjän tarpeiden kääntäminen ohjelmiston vaatimuksiksi, vaatimusten kokoaminen suunnitelmaksi, suunnitelman toteutus ohjelmoinnin avulla, testaus ja mahdollinen ohjelmiston operationaalisen käytön tarkistus [7]. Ei ole olemassa universaalia ohjelmistoprosessia, mutta prosessissa esiintyy aina jossakin muodossa neljä ohjelmistokehityksen peruseräpäätettä. Nämä ovat ohjelmiston vaatimusmäärittely, ohjelmistokehitys, ohjelmiston varmennus ja ohjelmiston päivitys. [6] Ohjelmistotuotanto on kallista ja aikaa vievää. Tämän takia tekoälyn mahdollisuuksista tehostaa ohjelmistotuotantoa ollaan erityisen kiinnostuneita. Singh ja Gautam [7] jakavat artikkelissaan ohjelmistokehitysprosessin neljään pääasialliseen vaiheeseen, joiden vaikutuksia ohjelmiston laatuun he tarkastelivat: ohjelmiston vaatimukset, ohjelmiston suunnittelu, ohjelmiston toteutus/ohjelmointi sekä ohjelmiston testaus, ja selvittävät mitä hyötyä näistä vaiheista on ohjelmistolle. Jokaisella vaiheista on vaikutuksensa lopullisen ohjelmiston laatuun. Tässä tekstissä generatiivisen tekoälyn vaikutuksia tarkastellaan näiden vaiheiden kautta. Vaiheet poikkeavat työnkuvaltaan paljon toisistaan, on siis olennaista tarkastella näitä erillisinä osa-alueinaan ja tekoälytyökalujen sovelluskohteita ja -tapoja näiden näkökulmista.

3.1 Ohjelmiston vaatimukset

Ohjelmiston vaatimusten määrittely on kriittinen vaihe, jossa pyritään ymmärtämään ja erittelemään kehitettävän ohjelmiston tarpeita ja rajoitteita. Virheet vaatimusten määrittelyssä johtavat vääjäämättä haasteisiin ohjelmiston suunnittelussa ja toteutuksessa. [6] Vaatimusmäärittely ohjaa koko kehitysprosessia ja kertoo mitä ohjelmiston tulee tehdä. Hyvin laaditut vaatimukset ovat selkeitä, mitattavia, testattavia ja jäljitettäviä, jolloin ne tukevat suunnittelua ja laadunvarmistusta. Vaatimusten tulee olla myös kattavia, johdonmukaisia ja muokattavia, jotta niitä voidaan ylläpitää ja päivittää ilman ristiriitoja. Laadukas vaatimustenmäärittely siis parantaa ohjelmiston laatua, luotettavuutta ja vaivatonta toteutettavuutta. [7]

3.2 Ohjelmiston suunnittelu

Ohjelmiston suunnittelu ja ohjelmointi voidaan erottaa toisistaan vaiheina, mutta esimerkiksi ketterässä kehityksessä ne ovat lomittaisia, eikä virallista suunnitteludokumentointia tehdä. Ketterässäkin kehityksessä suunnittelua kuitenkin tehdään ja tallennetaan epävirallisesti esimerkiksi valkotauluille tai ohjelmoijien muistiinpanoihin. [6] Vaiheiden toimenkuva on kuitenkin erilainen ja tekoälytyökaluja voidaan hyödyntää hyvin eri tavoilla suunnittelussa ja ohjelmoinnissa. Vaiheet erotellaan tässä työssä Singhin ja Gautamin [7] tutkimuksen mukaisissa raameissa.

Ohjelmiston suunnittelu on kuvaus ohjelmiston rakenteesta, toteutuksesta, datakaavioista, järjestelmän komponenttien käyttöliittymistä ja toisinaan käytetyistä algoritmeista [6]. Ohjelmiston suunnittelu on keskeistä ohjelmiston laadun kannalta, sillä sen avulla vaatimukset voidaan muuttaa toimivaksi kokonaisuudeksi. Suunnittelussa määritellään ohjelmiston arkkitehtuuri, komponentit ja rajapinnat, mikä mahdollistaa järjestelmän arvioinnin ja laadun varmistamisen. Hyvä suunnittelu parantaa ohjelmiston toiminnallisuutta, ylläpidettävyyttä, luotettavuutta, tehokkuutta ja muokattavuutta. Suunnitteluperiaatteet kuten arkkitehtuuri, modulaarisuus ja suunnittelumallit tukevat ohjelmiston uudelleenkäytettävyyttä, ymmärrettävyyttä ja joustavuutta. Tämä tekee ohjelmistosta vakaan ja helposti päivitettävän. [7]

3.3 Ohjelmiston toteutus

Ohjelmiston toteutus tai ohjelmointi on yksilöllinen vaihe, jota yleisesti hyväksytyt käytänteet eivät määrittele samalla tavalla kuin muita prosessin vaiheita. Toiset ohjelmoijat ottavat käsittelyyn ensimmäisenä ne komponentit, joita ymmärtävät parhaiten ja siirtyvät lopuksi niiden pariin, joista heillä on vielä opittavaa. Toiset ohjelmoijat taas ottavat käsittelyyn ensin heille vieraammat komponentit, jotta he voivat lopuksi kehittää ne komponentit, jotka ovat heille jo tuttuja. [6] Ohjelmointi muuttaa suunnitelman toimivaksi ohjelmaksi. Toteutusvaiheessa valitaan ohjelmointikielet ja käytännöt, jotka määrittävät muun muassa koodin ylläpidettävyyden, luotettavuuden ja testattavuuden. Selkeä, standardoitu ja helposti luettava koodi helpottaa ohjelmiston ymmärtämistä, muokkaamista ja jatkokehittämistä. Hyvin toteutettu koodi parantaa ohjelmiston vakautta, uudelleenkäytettävyyttä ja tehokkuutta. Vuorostaan huonolaatuinen koodi vaikeuttaa ylläpitoa ja heikentää luotettavuutta. [7]

3.4 Ohjelmiston testaus

Ohjelmiston testauksen tarkoituksena on näyttää ohjelman mukautuvan vaatimuksiin ja kohtaavan käyttäjän odotukset. Ohjelmistoa testataan pääasiallisesti simuloitua testidataa käyttäen. Ohjelmistolle voidaan tehdä myös muunlaisia tarkistuksia, esimerkiksi arvioimalla täyttääkö se kaikki käyttäjän tarpeet ja määritellyt vaatimukset. Vain pienet järjestelmät voidaan testata kokonaisina yksikköinä. Useimmat isommat järjestelmät tulee testata jokainen komponentti yksitellen ja tämän jälkeen näistä integroitu järjestelmä. [6] Ohjelmistotestaus on keskeinen osa laadun varmistamista, sillä sen avulla havaitaan virheet ja arvioidaan ohjelmiston toimivuutta. Testaus varmistaa, että ohjelmisto täyttää vaatimukset ja toimii luotettavasti eri olosuhteissa. Sen avulla parannetaan esimerkiksi ohjelmiston tehokkuutta, ylläpidettävyyttä ja käytettävyyttä. Testausmenetelmät, kuten yksikkö-, integraatio- ja järjestelmättestaus, mittaavat koodin kattavuutta ja laatua eri osalueilla. Hyvin suunniteltu testaus lisää ohjelmiston luotettavuutta, suorituskykyä ja varmuutta siitä, että tuote vastaa tarpeita. [7]

4. GENERATIIVISEN TEKOÄLYN ROOLI OHJELMISTOKEHITYKSESSÄ

Tekoälytyökalut ovat jo monilla yrityksillä käytössä. Esimerkiksi Slack:issa ChatGPT auttaa käyttäjiä työnkulun suunnittelussa. Freshworksin kaltaisissa yrityksissä tekoäly puolestaan tukee ohjelmoijia koodauksessa. Generatiivisen tekoälyn hyödyt ulottuvat myös esimerkiksi reaaliaikaiseen asiakaspalveluun, sisällön luomiseen bisnestoiminnalle, henkilöstöressurssien hallintaan, välittömään asiakaspalautteen saantiin, tiedonhankintaan eri tuotteista tai palveluista, tuotteiden hallintaan sekä digitaaliseen markkinointiin. [8] Generatiivista tekoälyä voidaan siis käyttää laajalti työn tehostamiseen myös itse ohjelmoinnin ulkopuolella. Ohjelmointiprojekteissa esiintyykin paljon tällaista työtä, kuten vaatimusten määrittelyä, käyttöliittymän suunnittelua sekä koodin testausta ja dokumentointia.

Tekoälytyökalujen hyödyntäminen vaatii taitoja käyttäjältä, kuten luodun koodin jalostamista [4] sekä kehotteen muodostamista. Esimerkiksi kehotteiden, joissa on sisäänrakennettu motivaatio, oikeanlainen rakenne tai annettu esimerkkejä, on havaittu vaikuttavan positiivisesti tuotettuihin tuloksiin [9]. Ohjelmistokehittäjän on tärkeää kasvavassa määrin olla tietoinen tekoälytyökalujen käytön yksityiskohdista, kuten kehotteen muotoilusta.

Generatiivinen tekoäly vähentää käyttäjän tarvetta tehdä rutiinitehtäviä [8], vaikka sen mahdollisuudet eivät rajoitu vain tehtävien automatisointiin [9]. Kirjoitustyössä tekoälytyökalu ChatGPT:n on havaittu vähentävän osaamiseroja, sillä se tukee erityisesti heikommin suoriutuvia käyttäjiä ja kaventaa näin tulosten välisiä eroja [10]. Generatiivisen tekoälyn erilaisia käyttömahdollisuuksia ja vaikutuksia tarkastellaan syvemmin ohjelmistokehityksen vaiheita käsittelevissä alaluvuissa.

Rico ja Öberg [11] selvittivät tutkimuksessaan yritysten johtoasemassa olevien henkilöiden olevan epävarmoja suurien kielimallien tuomisesta osaksi korkeamman tason tehtäviä. Johtajat tunnustivat kielimallien mahdollisuuksia luoda tai työstää niin vaatimuksia kuin suunnitteluehdotuksia sekä tehostaa projektin alkuvaiheita. Kielimallien käyttöönottoa kuitenkin haastoi organisaation varautuneisuus hyödyntää kielimalleja ohjelmointiin liittymättömissä tehtävissä ja huoli siitä, että epäselkeät vastaukset kasvattaisivat osakkeenomistajien väärinkäsityksiä. [11] Varautuneisuus uuteen teknologiaan voi vaikeuttaa sen käyttöönottoa. Generatiivisen tekoälyn parempi ymmärtäminen ja käytön hallinta voisi lieventää tätä varautuneisuutta ja ehkäistä ongelmia työssä.

4.1 Tekoäly ja vaatimukset

Generatiivisesta tekoälystä on hyötyä ohjelmiston vaatimuksia tarkastellessa. Tekoälytyökalut, kuten ChatGPT ja GitHub Copilot, voivat käydä läpi vaatimuksia, argumentoida mitkä kriteerit eivät täyty ja ehdottaa uutta versiota. Tekoälytyökalut voivat myös auttaa havaitsemaan epämääräistä terminologiaa ja ristiriitaisuuksia vaatimusmäärittelyissä, mikä tukee erityisesti aloittelijoita vaatimusten laadun parantamisessa. [9]

Krishna et al. [12] tutkivat suurten kielimallien käyttöä ohjelmiston vaatimusmäärittelyn dokumentaation tuottamisessa. Tutkimuksessa havaittiin, että kielimallien luoma dokumentaatio oli yleisesti hyvälaatuista ja verrattavissa aloittelijatason ohjelmistokehittäjän luomaan dokumentaatioon. Dokumentaatio oli johdonmukaista, selkeää ja tyypillisesti myös viimeisteltyä. [12] Tekoälytyökalun hyöty riippuu sekä valitun mallin kyvykkyydestä että käyttäjän osaamisesta. Jos huonosti valittu työkalu tuottaa enemmän korjattavaa kuin mitä dokumentaation laatiminen ohjelmoijalta vaatisi, sen käyttö ei ole mielekäästä.

Tutkimuksessa havaittiin myös selviä eroja mallien välillä. CodeLlama34b:n tuottama dokumentaatio oli edistyneen tietojenkäsittelyn opiskelijan tasoa, ja se vältti merkittäviä ongelmia, kuten epäjohdonmukaisuuksia, epätarkkuuksia ja poisti aikaa vievän rakenteellisen alustustyön. GPT-4:een perustuva ChatGPT tuotti puolestaan geneerisempiä, lyhyempiä ja vähemmän yksityiskohtaisia dokumentteja, vaikka sen yleinen laatu oli hyvällä tasolla. Vaatimusten varmistuksessa ja parantelussa ChatGPT suoriutui CodeLlama34b:tä paremmin, sillä se havaitsi ja korjasi enemmän puutteita vaatimuksissa. Oikein hyödynnettynä kielimalleilla laadittu vaatimusmäärittelydokumentaatio voidaan tuottaa moninkertaisesti nopeammin kuin aloittelevan ohjelmoijan tekemänä, mikä voi tarjota merkittävää etua vaatimusten määrittelyprosessissa. [12]

4.2 Tekoäly ja suunnittelu

Tekoälytyökalujen on havaittu vähentävän käyttäjän kognitiivista kuormitusta, parantavan suunnittelutyön luovuutta ja laatua silloin, kun ne integroidaan osaksi suunnittelu-prosessia. Pitkän aikavälin vaikutukset näihin tekijöihin ovat kuitenkin epäselviä, ja tekoälytyökaluja tulisi hyödyntää tavalla, joka tukee sekä syvällistä oppimista että ymmärrystä. [13] Tekoälytyökaluja kuten ChatGPT, Google Bard ja Microsoft Bing voidaan käyttää generoimaan ideoita ja aloitusvaiheessa työkaluja voidaan hyödyntää inspiraation lähteenä. Työkaluja kuten Uizard AI on saatavilla suunnitelman luomiseen. Uizard AI luo

useita käyttöliittymämalleja tekstisyötteen, kuvankaappauksien ja käsin piirrettyjen rautalankamallien pohjalta. Muita samanlaisia työkaluja ovat Prototype AI ja Figma AI:n liitännäiset. Näillä työkaluilla on huomattavaa potentiaalia mallinnusprosessin tehostamisessa, mutta ne eivät vielä kykene automatisoimaan prosessia kokonaan. [9] Tekoälytyökalu GitHub Copilotin suoritus ohjelmistomallinnuksessa on vaihtelevaa. Copilot voi tuottaa usein yksityiskohtaisia ja käyttökelpoisia sekvenssi- ja käyttötapauskaavioita, mutta luokka- ja tilakaavioista jää usein puuttumaan esimerkiksi keskeisiä luokkia ja suhteita. Paremmisakin kaavioissa esiintyy ongelmia väärin nimettyjen ja kehoitteessa täsmennettyjen elementtien muodossa. [14] Generatiivinen tekoäly ei siis pysty suorittamaan suunnittelullisia prosesseja alusta loppuun itsenäisesti, minkä vuoksi käyttäjän tehtäväksi jää virheiden etsiminen ja korjaaminen ongelmatilanteissa.

Suurin osa tekoälytyökalujen käytöstä suunnittelun tukena sijoittuu suunnittelun myöhempiin vaiheisiin, joissa pyritään kehittämään suunnittelullisia ratkaisuja. Alun ideointiin ja etsintään keskittyviä järjestelmiä on hyvin vähän. Suunnittelijat itse esittävät kiinnostusta järjestelmiä kohtaan, jotka avustaisivat näissä aikaisemmissa suunnittelutyön vaiheissa ongelmien löytämisessä, haasteiden kuvauksessa ja konseptuaalisten ideoiden luomisessa. Tekoäly on perinteisesti soveltunut paremmin selkeästi rajattuihin ja toistettaviin tehtäviin, kun taas suunnittelu vaatii monipuolista ajattelua ja intuitiota. Luonnolliset kielimallit voivat kuitenkin muuttaa tätä, sillä ne kykenevät käsittelemään jäsentymättömiä tehtäviä ja tukemaan päätöksentekoa suunnitteluprosessissa. [15]

4.3 Tekoäly ja toteutus

Tekoälytyökalujen käyttö on jo paikoitellen osa päivittäistä työtä yrityksissä, ja ne lievittävät toisteisen koodin kirjoittamisen vaivaa [11]. McKinseyn artikkelissa [16] todetaan, että ohjelmoijat ovat vähentäneet työaikaansa koodin kirjoittamiseen ja dokumentointiin liittyvissä tehtävissä noin puoleen. Taas refaktorointiin liittyvissä tehtävissä ohjelmoijat onnistuivat tekoälyn avulla vähentämään työajastaan kolmasosan. Monimutkaisemmissa tehtävissä tai kokemattomampien ohjelmoijien hyödyt olivat pienempiä, jopa alle 10 %. Tekoälytyökaluja käyttäneet ohjelmoijat onnistuivat kuitenkin 25–30 % todennäköisemmin suorittamaan annetut monimutkaisemmat tehtävät tavoiteajassa. Tekoälyä käyttäneet ohjelmoijat myös raportoivat korkeampaa tyytyväisyyttä, parempaa flow-tilaa ja palkitsevampia työkokemuksia. [16]

Ohjelmointikielen valinta on keskeinen tekijä, kun tarkastellaan tekoälytyökalujen hyödyntämistä ohjelmoinnissa. Eri kielten ja tekoälyn välinen vuorovaikutus voi johtaa siihen, että tekoälyn tuottamalla koodilla on kieleen sidottuja vahvuuksia ja heikkouksia. Sally Almasra ja Khaled Suwaisi [17] tutkivat artikkelissaan tekoälytyökalu ChatGPT-4:n kyvykkyyttä ratkaista algoritmisia ongelmiin sekä Java- että Python-kielillä. Tutkimuksen mukaan tekoälytyökalut suoriutuvat tehtävissä yleisesti hyvin, mutta niissä esiintyy merkittäviä laadullisia rajoitteita. Java-ratkaisut ovat usein suorituskyyvyltään nopeampia, kun taas Python-ratkaisut käyttävät muistia tehokkaammin. Vaikka tekoälytyökaluilla ongelmanratkaisun onnistumisprosentti on korkea, generoidun koodin dokumentaatiossa, poikkeusten käsittelyssä ja yleisessä rakenteellisessa siisteydessä, kuten käyttämättömien muuttujien esiintymisessä, on selviä puutteita. ChatGPT voikin tukea ohjelmointia tehokkaasti koodin luonnissa, mutta vaatii edelleen ihmisen valvontaa, jotta koodin huollettavuus, luotettavuus ja laatu voidaan varmistaa. [17]

Asha Rajbhoj et al. [18] käyttivät tutkimuksessaan meta-mallia, abstraktia rakennetta, joka koostuu selkeästi määritellyistä konsepteista kuten prosessi, aktiviteetti, parametri ja sääntö vaatimusmäärittelyssä, sekä käyttöliittymä-, palvelu- ja datakerroksista suunnittelussa. Mallin avulla luotiin rakenteeltaan vakiota kehotekaavoja ChatGPT:lle, joissa projektikohtaiset tiedot täytettiin automaattisesti, mikä mahdollisti järjestelmällisen ja toistettavan kehoteprosessi. Meta-mallista huolimatta nimitysten epäyhtenäisyydet, rakenteelliset puutteet ja paikoin tekniset virheet edellyttivät edelleen ihmisen tekemää tarkistusta ja korjausta. Tekoälyavusteisen ohjelmoinnin todettiin kuitenkin vauhdittavan ohjelmistokehitystä huomattavasti. [18] Vaikka kehotesuunnittelulla on positiivisia vaikutuksia luotuun koodiin, ei tämä siltikään takaa lopputulosta, joka ei vaatisi ihmisen tarkistusta ja korjausta.

Ohjelmiston dokumentointi on kallista ja aikaa vievää sekä luoda että ylläpitää. Useimmat ohjelmoijat eivät ole halukkaita dokumentoimaan, sillä tätä ei koeta tuotteliaana tai palkitsevana. Lisäksi dokumentaatio vanhenee ja vaatii jatkuvaa ylläpitoa ohjelman päivityessä. [19] Tekoälytyökaluja voidaankin käyttää apuna koodin dokumentoinnissa [9, 11]. Khan ja Uddin [19] tutkielmassa todetaan OpenAI:n ChatGPT-3:seen perustuvan tekoälytyökalu Codex:in suoriutuvan testeistä lähes yhtä hyvin oikean dokumentoinnin kanssa määrältään ja luettavuudeltaan. Joissakin tapauksissa Codex pystyi luomaan jopa oikeaa kattavamman dokumentaation. [19]

4.4 Tekoäly ja testaus

Tekoälytyökalut, kuten GitHub Copilot, ChatGPT ja Applitools, eivät ole vielä riittäviä generoimaan luotettavaa testauskoodia täysin automatisoidusti [9]. Kiinnostuksesta huolimatta synteettisen datan ja testien luominen kielimallien avulla on monessa yrityksessä toistaiseksi kokeellista, eikä vielä osa varsinaista tuotantokehitystä. Haasteeksi koetaan laadunvalvonnan korkea luottamuskyky. [11]

Testauksen käytännöt vaihtelevat huomattavasti, ja keskeistä on määrittää sekä mitä testataan että miltä pohjalta testit muodostetaan. Tämä kysymys korostuu erityisesti silloin, kun arvioidaan, miten generatiivista tekoälyä voidaan hyödyntää testien suunnittelussa ja tuottamisessa. Asha Rajbhoj et al. [18] havaitsivat tutkimuksessaan tekoälytyökalu ChatGPT:n tuottavan tyydyttävän taseisia testitapauksia, kun heillä oli käytössä kehotekaava, jonka avulla kehitysprosessi saatiin toistettavaksi ja systemaattiseksi. Testejä tuotettiin laajalti sovelluksen eri tasoille, kuten käyttöliittymään, palvelukerrokseen ja järjestelmätasolle, huomattavan vähäisellä manuaalisella korjaustarpeella. [18]

Käyttötapauskaaviot ovat tärkeä tapa välittää järjestelmän funktionaaliset vaatimukset ja interaktiot. Näiden kaavioiden kääntäminen suoritettaviksi testeiksi manuaalisesti on kuitenkin aikaa vievää, altista virheille ja harvoin riittävän kattavaa [20]. Generatiivinen tekoäly voi tarjota mahdollisuuden järjestelmälliseen ja tehokkaaseen testaamiseen, jos työkalut pystyvät esimerkiksi tällaisten käyttötapauskaavioiden pohjalta rakentamaan riittävän kattavia testejä.

Naimi et al. [20] esittävät lähestymistavan, jossa generatiivisen tekoälyn avulla automatisoidaan testien luominen, käyttäen syötteinä käyttötapauskaavioita. Lähestymistapa nopeuttaa merkittävästi testitapausten luontia, mikä voi nopeuttaa ohjelmistokehitystä ja tuotteen pääsyä markkinoille. Generatiivisen tekoälyn käyttö parantaa testauksen kattavuutta tuottamalla monipuolisia testitapauksia, jotka voisivat jäädä manuaalisissa menetelmissä huomaamatta. Menetelmän rajoitus on kuitenkin sen soveltuminen vain käyttötapauskaavioihin, tarjoten mahdollisesti suppean näkymän testattavasta järjestelmästä. Luotujen testien laatu riippuu tekoälymallin suorituskyvystä ja ylläpidosta. [20]

4.5 Yhteenveto

Seuraavissa taulukoissa esitellään keskeisimmät tulokset neljänestä luvusta, eli kuinka tekoälyä pystyttiin hyödyntämään eri ohjelmistokehityksen vaiheissa. Taulukko 1 käy läpi ohjelmistoprosessin alkuvaiheet, eli vaatimusten määrittelyyn ja suunnitteluun. Vaatimusten määrittelyssä tekoäly analysoi vaatimuksia, tunnistaa puutteita ja ehdottaa parannuksia. Sen tuottama dokumentaatio on usein selkeää ja käyttökelpoista. Suunnittelussa tekoäly vähentää kognitiivista kuormitusta, tukee luovuutta ja pystyy tuottamaan yksityiskohtaisia sekvenssi- ja käyttötapauskaavioita. Taulukko 2 keskittyy toteutus- ja testausvaiheisiin. Toteutuksessa tekoäly nopeuttaa koodin kirjoittamista, refaktorointia ja dokumentointia sekä parantaa suoriutumista monimutkaisemmissa tehtävissä. Testauksessa tekoälyllä voidaan luoda testitapauksia tehokkaasti ja kattavasti eri ohjelmistotasoille.

Taulukko 1. Tekoällyn hyödyntäminen ohjelmistoprosessin vaatimus- ja suunnitteluvaiheissa.

| Vaihe | Tekoällyn hyödyt |
|-------------|---|
| Vaatimukset | Tekoäly voi käydä läpi vaatimuksia, argumentoida mitkä kriteerit eivät täyty ja ehdottaa uutta versiota. [9] |
| | Tekoällyn luoma ohjelmiston vaatimusmäärittelydokumentaatio on yleisesti hyvälaatuista ja verrattavissa aloittelijatasen ohjelmistokehittäjän luomaan dokumentaatioon. [12] |
| Suunnittelu | Tekoälytyökalut vähentävät käyttäjän kognitiivista kuormitusta, parantavan suunnittelutyön luovuutta ja laatua suunnitteluprosessissa. [13] |
| | Tekoäly tuottaa usein yksityiskohtaisia ja käyttökelpoisia sekvenssi- ja käyttötapaus-kaavioita. [14] |

Taulukko 2 Tekoälyn hyödyntäminen ohjelmistoprosessin toteutus- ja testausvaiheissa.

| Vaihe | Tekoälyn hyödyt |
|----------|--|
| Toteutus | <p>Kun ohjelmoijilla on käytössä tekoälytyökaluja he onnistuvat vähentämään työaikaansa koodin kirjoittamiseen ja dokumentointiin liittyvissä tehtävissä noin puoleen. Taas refaktorointiin liittyvissä tehtävissä ohjelmoijat onnistuvat vähentämään työajastaan kolmasosan. Tekoälytyökaluja käyttävät ohjelmoijat onnistuvat myös 25–30 % todennäköisemmin suorittamaan annetut monimutkaisemmat tehtävät tavoiteajassa. Tekoälyä käyttävät ohjelmoijat myös raportoivat korkeampaa tyytyväisyyttä, parempaa flow-tilaa ja palkitsevampia työkokemuksia. [16]</p> |
| | <p>Tekoälyn tuottama dokumentointi suoriutuu alkuperäisen kanssa lähes yhtä hyvin määrältään ja luettavuudeltaan. Joissakin tapauksissa tekoäly pystyi luomaan jopa alkuperäistä kattavamman dokumentaation. [19]</p> |
| Testaus | <p>Tekoäly voi tuottaa tyydyttäviä testitapauksia kehotekaavan avulla. Testejä voidaan tuottaa laajalti sovelluksen eri tasoille, kuten käyttöliittymään, palvelukerrokseen ja järjestelmätasolle. [18]</p> |
| | <p>Tekoäly voi luoda testejä käyttäen syötteenä käyttötapauskaavioita. Lähestymistapa nopeuttaa merkittävästi testitapausten luontia ja tekoälyn käyttö parantaa testauksen kattavuutta tuottamalla monipuolisia testitapauksia, jotka voisivat jäädä manuaalisissa menetelmissä huomaamatta. [20]</p> |

5. GENERATIIVISEN TEKOÄLYN HAASTEET

Generatiivisen tekoälyn tuomien ohjelmistokehityksen mahdollisuuksien rinnalla kulkevat sen tuomat haasteet. Immateriaalioikeudet, tietoturva ja tuotosten luotettavuus voivat olla todella vaikeita asioita taata, jos tekoälytyökaluja käytetään tai kehitetään ajattelemattomasti.

Kielimallien hyödyntäminen koodin tuottamisessa herättää erityisesti immateriaalioikeuksiin liittyviä huolia, kuten kolmansien osapuolten koodin luvaton käyttö tai lisenssi-oikeuksien rikkominen. Samalla syntyy haasteita tuotetun koodin luotettavuuden ja tietoturvan varmistamisessa. On myös esitetty kysymyksiä, siitä miten tekoälyn luoman koodin ylläpito onnistuu tulevaisuudessa, jos sen alkuperä, suunnitteluperiaatteet tai rakenteet ovat epäselviä ohjelmoijille. [11]

Koska generatiivinen tekoäly perustuu laajoihin aineistoihin ja osin ohjaamattomaan oppimiseen, järjestelmät voivat tuottaa vastauksia, jotka jäljittelevät ihmisen kirjoittamaa kieltä, mutta sisältävät myös virheitä tai vinoumia. Tämä lisää riskiä, että tekoälyä voidaan käyttää tietoisesti tai tahattomasti manipuloiviin tarkoituksiin. On havaittu tapauksia, joissa harhaanjohtavat tai vinoutuneet aineistot ovat vaikuttaneet tuotettuun sisältöön. Lisäksi asiakastietojen yksityisyys herättää huolta yrityksissä, sillä tietojen kerääminen esimerkiksi evästeiden avulla voi vaikuttaa negatiivisesti organisaatioiden liiketoimintaan ja julkiskuvaan. [8] Siksi on tärkeää arvioida tekoälytyökalujen eettisiä ja tietoturvaan liittyviä vaikutuksia sekä suunnitella, miten mahdollisia haittoja ehkäistään, jos tekoälyä aiotaan hyödyntää ohjelmistoprosessissa.

Tekoälytyökalut vaativat ohjelmoinnissa edelleen ihmisen valvontaa, sillä ne voivat tuottaa virheellisiä tai epätarkkoja ehdotuksia. Generoidun koodin luotettavuus ei toistaiseksi riitä siihen, että sitä voisi käyttää ilman asiantuntevaa tarkistusta [17], eikä edes kehotesuunnittelulla saavuteta täydellistä varmuutta [18]. Työkalujen rajallinen kontekstinymmärrys edellyttää ohjelmoijan panosta, jotta tuotettu koodi vastaa todellisia tarpeita. Monimutkaiset, monirakenteiset ja vahvasti integroidut ohjelmistotehtävät hyötyvät tekoälystä vähemmän kuin toisteiset ja yksinkertaiset tehtävät. [16]

6. YHTEENVETO

Tutkimuksen perusteella generatiivinen tekoäly on vakiinnuttanut asemansa ohjelmistokehityksessä, erityisesti ohjelmoinnin osa-alueella, jossa suuret kielimallit voivat tukea koodin tuottamista, dokumentaatiota, virheenkorjausta sekä testauksen automatisointia. Tekoäly vähentää rutiiniväistöä ja vapauttaa kehittäjiä keskittymään monimutkaisempiin tehtäviin, mikä tehostaa työnkulkua ja parantaa tuottavuutta. Vaikka tekoälyn soveltuvuus vaatimusten määrittelyyn, suunnitteluun ja testauksen syvällisempään tukeen ovat kehityksessä, eivät ne ole vielä yhtä kehittyneitä kuin toteutukseen liittyvät ratkaisut.

Tulokset korostavat myös osaamistarpeiden muuttumista. Tekoälyn tehokas hyödyntäminen edellyttää kehittäjiltä uusia taitoja, kuten kykyä laatia täsmällisiä ja tarkoituksenmukaisia kehoitteita sekä arvioida tekoälyn tuottamien tulosten laatua kriittisesti. Nämä taidot vaikuttava siihen, miten luotettavia, käyttökelpoisia ja ylläpidettäviä tuotetut ohjelmistot ovat.

Generatiivisen tekoälyn käyttöön liittyy kuitenkin merkittäviä haasteita. Tietoturva, immateriaalioikeudet, mahdollisesti vinoutunut opetusdata sekä tuotetun koodin luotettavuus muodostavat riskejä, jotka on huomioitava ohjelmistokehityksessä. Suunnitteluvaiheessa tekoälyn vaikutusten arviointi voi olla haastavaa, koska suunnittelu on luova ja osin subjektiivinen prosessi. Tekoälyn aiheuttama kognitiivisen kuorman keventyminen on selvästi positiivista, mutta käyttöliittymien visuaalisen laadun kaltaisia tekijöitä on vaikeampi arvioida objektiivisesti. Vartenotettava haaste voi myös esiintyä ohjelmistojen päivityksessä ja ylläpidossa, jos ohjelmasta suuria osia on tuotettu tekoälyllä ja ratkaisujen tarkoituksperiä tai suunnitteluperiaatteita voi olla haastavaa selvittää.

Tutkimuksen aikajänne osoittaa, että generatiivisen tekoälyn nopea kehitys muodostaa haasteen tutkimukselle. Esimerkiksi GPT-5 -malli, joka on jo saatavilla käyttäjille, ei vielä näy tarkastelluissa tutkimuksissa. Kirjallisuus voi siis vanhentua nopeasti, ja tutkimusten ajantasaisuuden varmistaminen edellyttää jatkuvaa seuranta.

Kokonaisuutena generatiivinen tekoäly tarjoaa huomattavaa potentiaalia ohjelmistokehityksen tehostamiseen ja laadun parantamiseen. Samalla sen vastuullinen ja tehokas hyödyntäminen vaatii huolellista suunnittelua, eettistä harkintaa ja kehittäjien osaamisen jatkuvaa päivittämistä.

LÄHTEET

- [1] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekkilä, D. Doermann, "Future of software development with generative AI," *Automated Software Engineering*, vol. 31, no. 1, p. 26, May 2024, doi: <https://doi.org/10.1007/s10515-024-00426-z>.
- [2] NVIDIA, "What is Generative AI?," *NVIDIA Glossary*, [Online]. Available: <https://www.nvidia.com/en-us/glossary/generative-ai/>. [Accessed: 18-Nov-2025].
- [3] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, "Large Language Models for Software Engineering: Survey and Open Problems," in *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, Melbourne, Australia: IEEE, May 2023, pp. 31–53. doi: [10.1109/ICSE-FoSE59343.2023.00008](https://doi.org/10.1109/ICSE-FoSE59343.2023.00008).
- [4] A. Bahi, J. Gharib, Y. Gahi, "Integrating Generative AI for Advancing Agile Software Development and Mitigating Project Management Challenges," *IJACSA*, vol. 15, no. 3, 2024, doi: [10.14569/IJACSA.2024.0150306](https://doi.org/10.14569/IJACSA.2024.0150306).
- [5] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, I. Gazit, "Taking Flight with Copilot," *Commun. ACM*, vol. 66, no. 6, pp. 56–62, Jun. 2023, doi: [10.1145/3589996](https://doi.org/10.1145/3589996).
- [6] I. Sommerville, *Software engineering*, Tenth edition. Pearson, 2016.
- [7] B. Singh & S. Gautam. "The Impact of Software Development Process on Software Quality: A Review." *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*, IEEE, December 2016, 666–72. <https://doi.org/10.1109/CICN.2016.137>.
- [8] N. P. Rana, R. Pillai, B. Sivathanu, N. Malik, "Assessing the nexus of Generative AI adoption, ethical considerations and organizational performance," *Technovation*, vol. 135, p. 103064, July 2024, doi: [10.1016/j.technovation.2024.103064](https://doi.org/10.1016/j.technovation.2024.103064).
- [9] M. Fischer & C. Lanquillon, "Evaluation of Generative AI-Assisted Software Design and Engineering: A User-Centered Approach," in *Artificial Intelligence in HCI*, H. Degen and S. Ntoa, Eds., in *Lecture Notes in Computer Science*, vol. 14734, Cham: Springer Nature Switzerland, 2024, pp. 31–47. doi: [10.1007/978-3-031-60606-9_3](https://doi.org/10.1007/978-3-031-60606-9_3).
- [10] S. Noy & W. Zhang, "Experimental evidence on the productivity effects of generative artificial intelligence". Jul 2023. Vol 381, Issue 6654. pp. 187–192. doi: [10.1126/science.adh2586](https://doi.org/10.1126/science.adh2586)

- [11] S. Rico & L.-M. Öberg, “Challenges and Opportunities for Generative AI in Software Engineering: A Managerial View,” in *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, Clarion Hotel Trondheim Trondheim Norway: ACM, June 2025, pp. 1338–1344. doi: [10.1145/3696630.3728718](https://doi.org/10.1145/3696630.3728718).
- [12] M. Krishna, B. Gaur, A. Verma, P. Jalote, “Using LLMs in Software Requirements Specifications: An Empirical Evaluation,” in *2024 IEEE 32nd International Requirements Engineering Conference (RE)*, Reykjavik, Iceland: IEEE, June 2024, pp. 475–483. doi: [10.1109/RE59067.2024.00056](https://doi.org/10.1109/RE59067.2024.00056).
- [13] T. Chandrasekera, Z. Hosseini, U. Perera, “Can artificial intelligence support creativity in early design processes?,” *International Journal of Architectural Computing*, vol. 23, no. 1, pp. 122–136, Mar. 2025, doi: [10.1177/14780771241254637](https://doi.org/10.1177/14780771241254637).
- [14] R. Ramachandran, “Empowering Software Architects with Artificial Intelligence: Analyzing GitHub Copilot’s Role in Modern Architecture Design,” in *2025 7th International Conference on Software Engineering and Computer Science (CSECS)*, Taicang, China: IEEE, Mar. 2025, pp. 1–9. doi: [10.1109/CSECS64665.2025.11009821](https://doi.org/10.1109/CSECS64665.2025.11009821).
- [15] S. Lee, M. Law, G. Hoffman, “When and How to Use AI in the Design Process? Implications for Human-AI Design Collaboration,” *International Journal of Human–Computer Interaction*, vol. 41, no. 2, pp. 1569–1584, Jan. 2025, doi: [10.1080/10447318.2024.2353451](https://doi.org/10.1080/10447318.2024.2353451).
- [16] B. Karaci Deniz, C. Gnanasambandam, M. Harrysson, A. Hussin and S. Srivastava, “Unleashing developer productivity with generative AI,” McKinsey Digital, 27 June 2023. [Online]. Available: <https://www.mckinsey.com/capabilities/tech-and-ai/our-insights/unleashing-developer-productivity-with-generative-ai/>
- [17] S. Almanasra & K. Suwais, “Analysis of ChatGPT-Generated Codes Across Multiple Programming Languages,” *IEEE Access*, vol. 13, pp. 23580–23596, 2025, doi: [10.1109/ACCESS.2025.3538050](https://doi.org/10.1109/ACCESS.2025.3538050).
- [18] A. Rajbhoj, A. Somase, P. Kulkarni, V. Kulkarni, “Accelerating Software Development Using Generative AI: ChatGPT Case Study,” in *Proceedings of the 17th Innovations in Software Engineering Conference*, Bangalore India: ACM, Feb. 2024, pp. 1–11. doi: [10.1145/3641399.3641403](https://doi.org/10.1145/3641399.3641403).

[19] J. Y. Khan & G. Uddin, "Automatic Code Documentation Generation Using GPT-3," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, Rochester MI USA: ACM, Oct. 2022, pp. 1–6. doi: [10.1145/3551349.3559548](https://doi.org/10.1145/3551349.3559548).

[20] L. Naimi, E. M. Bouziane, M. Manaouch, A. Jakimi, "A new approach for automatic test case generation from use case diagram using LLMs and prompt engineering," in *2024 International Conference on Circuit, Systems and Communication (ICCSC)*, Fes, Morocco: IEEE, June 2024, pp. 1–5. doi: [10.1109/ICCSC62074.2024.10616548](https://doi.org/10.1109/ICCSC62074.2024.10616548).