TURUN YLIOPISTO
Kauppakorkeakoulu

# Tekoälyavusteinen ohjelmistokehitys Suomen terveydenhuollossa

Tietojärjestelmätiede, Markkinoinnin ja arvoketjujen johtamisen laitos

Kandidaatintutkielma

Laatija:
Aino-Kaisa Kärkkäinen

Ohjaaja:
FT Samuli Laato

12.12.2025
Turku

Opiskelijan lausunto tekoälyn käytöstä tähän tutkielmaan liittyen:

☐ **En ole käyttänyt tekoälyä hyödyntäviä työkaluja** tätä tutkielmaa kirjoittaessani.

☒ **Olen käyttänyt tekoälyä hyödyntäviä työkaluja** tätä tutkielmaa kirjoittaessani. Tämä käyttö on dokumentoitu tutkielman liitteessä. Vakuutan, että tekoälyä käytettiin yliopiston ohjeistuksen mukaisella tavalla.

Turun yliopiston laatujärjestelmän mukaisesti tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -järjestelmällä.

Kandidaatintutkielma

**Oppiaine**: Tietojärjestelmätiede
**Tekijä**: Aino-Kaisa Kärkkäinen
**Otsikko**: Tekoälyavusteinen ohjelmistokehitys Suomen terveydenhuollossa
**Ohjaaja**: FT Samuli Laato
**Sivumäärä**: 33 sivua + liitteet 1 sivu
**Päivämäärä**: 12.12.2025

## Tiivistelmä

Terveydenhuollon ohjelmistokehitystä määrittää jännite kasvavien tehokkuuspaineiden ja ehdottomien turvallisuusvaatimusten välillä. Tämän kandidaatintutkielman taustalla on tarve selvittää, miten generatiivinen tekoäly voi vastata näihin haasteisiin vaarantamatta potilasturvallisuutta. Tutkielmassa etsitään kirjallisuuskatsauksen avulla vastauksia kahteen tutkimuskysymykseen: mitä erityispiirteitä ohjelmistokehitykseen liittyy terveydenhuollon toimintaympäristössä, etenkin Suomessa, ja kuinka generatiivisella tekoälyllä tuotettua koodia voidaan hyödyntää tässä kontekstissa.

Tutkimuksessa käytetty aineisto koostuu alan aiemmasta tutkimuskirjallisuudesta, viranomaislähteistä sekä lainsäädännöstä, jotka määrittelevät Suomen terveydenhuollon tietojärjestelmien reunaehdot. Tavoitteena oli muodostaa kokonaiskuva tekoälyn mahdollisuuksista ja rajoitteista säännellyssä ympäristössä.

Kirjallisuuskatsauksen tulokset osoittivat, että Suomen terveydenhuollon ohjelmistokehitystä määrittävät tietoturvan, tietosuojan, toiminnallisuuden ja yhteentoimivuuden toteutumisen vaatimukset. Merkittäviä haasteita ohjelmistoissa aiheuttavat tällä hetkellä legacy-järjestelmät, jotka sitovat resursseja ja vaikeuttavat järjestelmien integraatiota. Generatiivinen tekoäly tarjoaa ratkaisuja näihin ongelmiin toimimalla kehittäjän apuvälineenä. Se voi nopeuttaa dokumentaatiota, tehostaa testausta luomalla testitapauksia sekä tukea vanhan koodin modernisointia. Lisäksi generatiivinen tekoäly voi parantaa kommunikaatiota teknisten asiantuntijoiden ja hoitohenkilökunnan välillä toimimalla tulkkina vaatimusmäärittelyssä.

GenAI:n hyödyntämiseen liittyy kuitenkin merkittäviä riskejä, kuten mallien taipumus hallusinoida eli tuottaa virheellistä tietoa sekä päätöksenteon läpinäkymättömyys, joka on ristiriidassa lääketieteellisen etiikan selitettävyysvaatimusten kanssa. Johtopäätöksenä todetaan, että tekoälyavusteinen koodin generointi on potentiaalinen menetelmä terveydenhuollon ohjelmistokehityksen eri vaiheiden tehostamiseen, kunhan vältytään uuden läpinäkymätöömyysongelman syntymiseltä. Tämä edellyttää ehdotonta kehittäjän valvontaa, jossa vastuu koodin laadusta ja turvallisuudesta säilyy ihmisellä.

**Avainsanat:** generatiivinen tekoäly, ohjelmistokehitys, terveydenhuollon tietojärjestelmät, suuret kielimallit, vaatimusmäärittely, legacy-modernisointi

# SISÄLLYS


1 Johdanto 7

2 Ohjelmistokehitys terveydenhuollossa 9

2.1 Terveydenhuollon ohjelmistojen keskeiset vaatimukset Suomessa 9

2.2 Käytettävyys ja loppukäyttäjien osallistaminen kehitykseen 10

2.3 Ketterien menetelmien soveltaminen 11

2.4 Terveydenhuollon ohjelmistokehityksen haasteet ja mahdollisuudet 12

2.4.1 Teknisen velan ja monimutkaisen toimintaympäristön haasteet 12

2.4.2 Standardisaation ja uusien teknologioiden tarjoamat mahdollisuudet 14

3 Generatiivisella tekoälyllä tuotetun koodin hyödyntäminen terveydenhuollon ohjelmistokehityksessä 16

3.1 Tekoälyavusteiset koodigeneraattorit ja niiden toimintaperiaate 16

3.2 Generatiivisen tekoälyn hyödyt terveydenhuollon ohjelmistokehitysprosessissa 19

3.3 Käyttötapaukset terveydenhuollon kontekstissa 22

3.4 Tekoälyllä generoidun koodin rajoitukset ja riskitekijät 24

4 Yhteenveto ja johtopäätökset 26

Lähteet 29

5 Liitteet 34

Liite 1 Selvitys tekoälyn käytöstä 34

## KUVIOT

KUVIO 1 GENERATIIVISEN TEKOÄLYN HYÖDYT OHJELMISTOKEHITYKSEN ERI VAIHEISSA 22

## TAULUKOT

TAULUKKO 1. SITTIG YM. (2020) ESITTÄMÄT TERVEYDENHUOLLON TIETOJÄRJESTELMIEN YHDEKSÄN LYHYEN AIKAVÄLIN HAASTETTA 14
TAULUKKO 2. KESKEISET TEKOÄLYPOHJAISET KOODIGENERAATTORIT JA NIIDEN OMINAISUUDET 18

# 1 Johdanto

Automaattiseen koodin generointiin kohdistuu nykyään merkittäviä odotuksia, sillä sen nähdään tarjoavan ratkaisun ohjelmistotuotannon kasvaviin tehokkuuspaineisiin (Odeh ym., 2024). Erityisesti generatiiviseen tekoälyyn (GenAI) perustuvat koodigeneraattorit ovat osoittaneet potentiaalia ohjelmistokehityksen tehostamiseksi niiden nopeuden ja kustannustehokkuuden vuoksi. Tarve uusille ratkaisuille on ilmeinen terveydenhuollossa, jossa tietojärjestelmät muodostavat nykyaikaisen potilashoidon perustan. Suomessa terveydenhuollossa painetta lisäävät väestön ikääntyminen ja palveluntarpeen kasvu, joiden aiheuttamaa resurssivajetta pyritään paikkaamaan digitalisaation ja uusien teknologioiden avulla (Sosiaali- ja terveysministeriö, 2023). Ala on taloudellisesti merkittävä myös globaalisti ja kasvaa voimakkaasti teknologian kehittyessä. Esimerkiksi terveydenhuollon esineiden internetin (engl. Internet of Things, IoT) markkinan liikevaihdon on arvioitu nousevan maailmanlaajuisesti noin 84 miljardista Yhdysvaltain dollarista vuonna 2024 yli 135 miljardiin vuoteen 2029 mennessä (Statista, 2025). Tämä voimakas kasvu luo painetta kehittää ohjelmistoja entistä nopeammin ja kustannustehokkaammin, mutta terveydenhuollon toimintaympäristö asettaa kehitystyölle poikkeuksellisen tiukat reunaehdot.

Ohjelmoinnin tueksi kehitetyt tekoälytyökalut eivät vielä nykyisessä kehitysvaiheessaan korvaa ohjelmoinnin ammattilaisia, mutta sekä Ebertin ja Louridaksen (2023) että Odehin ym. (2024) mukaan generatiivista tekoälyä voidaan hyödyntää erityisen hyvin aikaa vaativien toistuvien tehtävien suorittamisessa. Odeh ym. (2024) toteavat, että tämän kehityksen myötä ohjelmistokehityssyklit nopeutuvat, ja ohjelmistokehittäjät voivat keskittyä vaativimpiin tehtäviin ja ongelmanratkaisuun. Heidän mukaansa tekoälyn käyttöön liittyy myös haasteita ja rajoituksia muun muassa eettisten kysymysten, skaalautuvuuden ja datan laadun osalta, joita käsitellään tutkielman seuraavissa luvuissa.

Oikein toimiessaan tietojärjestelmät tarjoavat turvallisen ja nopean tavan säilyttää ja käsitellä suurta määrää potilastietoja. Suomessa sosiaali- ja terveydenhuollon järjestelmille on asetettu vaatimukset laissa (Laki sosiaali- ja terveydenhuollon asiakastietojen sähköisestä käsittelystä 784/2021). Lainsäädäntö ja alan standardit edellyttävät, että järjestelmät ovat yhteentoimivia, toiminnallisesti tarkoituksenmukaisia sekä tietoturvan ja tietosuojan osalta riittävän vahvoja (Rytkönen ym., 2022). Sääntelyn noudattamisen valvonnasta vastaa Suomessa Valvira.

Tekoälyn tuoma tehokkuus ja terveydenhuollon alan vaatimukset ovat kuitenkin osittain ristiriidassa keskenään. Vaikka luonnollisella kielellä tapahtuva nopea sovelluskehitys (engl. vibe coding) voi tuottaa tuloksia hetkissä, Abgrall ym. (2025) varoittavat sen riskeistä: nopeasti generoitu koodi voi

olla haurasta tai sisältää tietoturva-aukkoja, jos prosessissa ei ole mukana ihmistä varmistamassa laatua. Lisäksi vanhentuneiden järjestelmien (engl. legacy systems) ylläpito ja päivittäminen vievät terveydenhuollon ohjelmistokehityksessä merkittävän määrän resursseja (Kruse ym., 2016), ja generatiivisen tekoälyn hyödyntäminen niiden modernisoinnissa vaatii tasapainoilua innovaation ja riskienhallinnan välillä. Tämän tasapainon löytäminen on kriittistä, jotta terveydenhuollon ohjelmistokehitys pystyy vastaamaan kasvaviin tehokkuuspaineisiin vaarantamatta potilasturvallisuutta tai säädöstenmukaisuutta.

Tässä kandidaatintutkielmassa tarkastellaan mahdollisuuksia tehostaa koodin tuottamista terveydenhuollon ohjelmistokehitysprosesseissa käyttäen generatiivista tekoälyä sekä sitä, kuinka se voidaan tehdä tehokkaasti ja turvallisesti. Tutkielma vastaa seuraaviin tutkimuskysymyksiin:

1. Mitä erityispiirteitä ohjelmistokehitykseen liittyy terveydenhuollossa?
2. Kuinka generatiivisella tekoälyllä tuotettua koodia voidaan käyttää terveydenhuollon ohjelmistokehityksessä?

Tutkimuskysymyksiin vastataan kuvailevalla kirjallisuuskatsauksella. Vastaukset muodostetaan syntetisoimalla tietoa alan aiemmasta tutkimuskirjallisuudesta ja viranomaislähteistä. Johdannon jälkeen luvussa 2 vastataan ensimmäiseen tutkimuskysymykseen jäsentelemällä kohteena olevan alan erityispiirteitä sekä sääntelykehys ohjelmistokehityksessä. Alueellisia erityispiirteitä tarkastellaan etenkin Suomen kontekstissa. Luvussa 3 vastataan toiseen tutkimuskysymykseen esittelemällä generatiivisen tekoälyn käyttökohteita, hyötyjä sekä tunnistettuja riskejä alan ohjelmistokehitysprojekteissa. Työ on rajattu tarkastelemaan generatiivisen tekoälyn käyttöä nimenomaan sovelluskehityksen ja koodin tuottamisen näkökulmasta, jättäen hoidolliset tekoälysovellukset, kuten kuvantamisdiagnostiikan, tarkastelun ulkopuolelle, elleivät ne liity suoraan ohjelmistokehitysprosessiin.

# 2 Ohjelmistokehitys terveydenhuollossa

## 2.1 Terveydenhuollon ohjelmistojen keskeiset vaatimukset Suomessa

Terveydenhuollon alalla laki (Laki sosiaali- ja terveydenhuollon asiakastietojen sähköisestä käsittelystä 784/2021) sekä alan vakiintunut käytäntö edellyttävät, että käytettävät tietojärjestelmät täyttävät Kansaneläkelaitoksen (Kela) asettamat vaatimukset yhteentoimivuudesta, toiminnallisesta tarkoituksenmukaisuudesta sekä tietoturvan ja tietosuojan vahvuudesta. Rytkösen ym. (2022) mukaan näiden vaatimusten tarkoituksena on se, että tietojärjestelmät mahdollistavat sosiaali- ja terveydenhuollon päätehtävän, eli kansalaisten hyvinvoinnin ja terveyden edistämisen sekä sairauksien hoitamisen parhaalla mahdollisella tavalla.

Yhteentoimivuuden osalta vaatimuksena on, että potilastieto tuotetaan ja tallennetaan niin, että tieto voidaan hakea ja näyttää myös toisella tietojärjestelmällä. Tietojärjestelmien yhteentoimivuuden testaa ja todentaa Kela. (Valvira, ei pvm.) Yhteentoimivuus on olennainen osa terveydenhuollon tietojärjestelmiä, sillä potilastiedot on kyettävä siirtämään turvallisesti eri järjestelmien ja toimijoiden välillä. Ilman riittävää yhteentoimivuutta tietojen siirtyminen eri organisaatioiden välillä voi viivästyä tai estyä kokonaan, mikä voi vaarantaa potilasturvallisuuden ja heikentää hoidon laatua. Yhteentoimivuuden varmistamiseksi tietojärjestelmien on noudatettava kansallisia ja kansainvälisiä standardeja, kuten HL7 FHIR (engl. Fast Healthcare Interoperability Resources) -protokollaa, joka mahdollistaa tietojen rakenteisen ja standardoidun käsittelyn. (Saripalle ym., 2019.) Suomessa Kantapalvelu on keskeinen yhteentoimivuuden mahdollistaja, ja kaikkien terveydenhuollon tietojärjestelmien tulee pystyä hyödyntämään sen tarjoamia palveluita (Laki sosiaali- ja terveydenhuollon asiakastietojen sähköisestä käsittelystä 784/2021).

Toiminnallisuus kattaa tietojärjestelmien käytettävyys- ja suorituskykyvaatimukset. Järjestelmien on tuettava kliinistä työtä tarjoamalla sujuvat käyttöliittymät, tehokkaat hakutoiminnot sekä päätöksenteon tueksi kehitetyt algoritmit. Lisäksi järjestelmien tulee mukautua terveydenhuollon lainsäädännön ja toimintaympäristön muutoksiin ilman merkittäviä käyttökatkoja. (Laki sosiaali- ja terveydenhuollon asiakastietojen käsittelystä, 703/2023.) Heikko käytettävyys voi johtaa lisääntyneeseen työkuormaan ja virheiden syntymiseen, mikä puolestaan voi vaikuttaa negatiivisesti potilasturvallisuuteen (Kaipio ym., 2017).

Tietoturva on kriittinen vaatimus, sillä terveydenhuollon tietojärjestelmät käsittelevät arkaluonteisia henkilötietoja. Järjestelmien on täytettävä tietoturvaa koskevat tekniset ja hallinnolliset vaatimukset, jotta tietojen käsittelyn luottamuksellisuus, eheys ja käytettävyys toteutuvat. Luottamuksellisuuden

vaatimuksella tarkoitetaan sitä, että arkaluontoiset tiedot ovat ainoastaan niiden henkilöiden nähtävillä, joilla on oikeus ne nähdä. Eheys toteutuu silloin, kun tietoja voivat muuttaa ainoastaan ne henkilöt, joilla on siihen oikeus. Tietojen tulisi olla aina ajantasaisia ja ristiriidattomia. Käytettävyydellä viitataan siihen, että tiedot ovat käytettävissä juuri silloin, kun siihen on tarve. (Valvira, ei pvm.) Tietoturvauhat, kuten kyberhyökkäykset ja tietovuodot, ovat kasvava riski terveydenhuollossa, minkä vuoksi järjestelmien on oltava jatkuvan seurannan ja päivitysten kohteena. Tietoturvapuutteet voivat vaarantaa sekä potilaiden yksityisyyden että organisaatioiden luotettavuuden. (ENISA, 2023.)

Tietosuoja liittyy kiinteästi tietoturvaan, mutta siinä painottuu erityisesti potilastietojen käsittelyn lainmukaisuus ja potilaiden oikeudet. EU:n yleinen tietosuoja-asetus (engl. general data protection directive, GDPR) asettaa vaatimuksia sille, miten henkilötietoja saa käsitellä ja säilyttää. Terveydenhuollon tietojärjestelmien on varmistettava, että tietojen käsittely perustuu laillisiin perusteisiin, kuten potilaan suostumukseen tai terveydenhuollon lakisääteisiin velvoitteisiin. Lisäksi järjestelmien tulee mahdollistaa potilaiden oikeuksien toteutuminen, kuten pääsy omiin tietoihin ja niiden korjaaminen tarvittaessa. (Euroopan unionin virallinen lehti, 2016.)

Edellä mainitut vaatimukset muodostavat terveydenhuollon tietojärjestelmien perustan ja vaikuttavat suoraan niiden kykyyn tukea turvallista ja tehokasta potilashoitoa. Vaatimusten noudattaminen edellyttää jatkuvaa kehitystyötä ja yhteistyötä niin ohjelmistokehittäjien, terveydenhuollon ammattilaisten kuin viranomaisten välillä.

## 2.2 Käytettävyys ja loppukäyttäjien osallistaminen kehitykseen

Käytettävyys on keskeinen laadullinen ominaisuus lähes kaikissa tietojärjestelmissä, joilla on loppukäyttäjiä, mukaan lukien terveydenhuollon tietojärjestelmissä. Kansainvälisen ISO 9241 -standardin määritelmän mukaan käytettävyydellä tarkoitetaan sitä, kuinka hyvin tietyt käyttäjät voivat tietyssä käyttöympäristössä saavuttaa tavoitteensa tehokkaasti, tuloksellisesti ja tyytyväisinä järjestelmää käyttäen. Yenin ja Bakkenin (2012) mukaan tämä tarkoittaa terveydenhuollon kontekstissa sujuvaa työnkulkua, jossa järjestelmä tukee ammattilaisen työtä eikä vie huomiota pois potilaasta. Heikko käytettävyys ei ole ainoastaan mukavuuskysymys, vaan se voi johtaa tehottomuuteen, lisääntyneisiin kustannuksiin ja pahimmillaan vaaratilanteisiin potilastyössä (Yen & Bakken, 2012).

Johnsonin ym. (2005) mukaan järjestelmien kehittämisessä on pitkään ollut haasteena teknologian ja kliinisen työn välinen kuilu, ja ongelmat kehittämisessä juontavat usein juurensa siihen, että suunnittelijan mielikuva järjestelmästä ei vastaa käyttäjän mielikuvaa todellisesta työtehtävästä.

Käyttäjäkeskeisyyden sivuuttaminen voi johtaa kalliisiin epäonnistumisiin, sillä virheiden korjaaminen kehitysvaiheessa on Johnsonin ym. (2005) arvion mukaan 10 kertaa kalliimpaa, ja järjestelmän julkaisun jälkeen arviolta jopa 100 kertaa kalliimpaa kuin suunnitteluvaiheessa. Mikäli käyttäjien todellisia tarpeita ja työprosesseja ei oteta huomioon, vaarana on kallis epäonnistuminen, jossa järjestelmä ei sovellu tarkoitukseensa eikä sitä saada hyödynnettyä tehokkaasti tai lainkaan.

Kushniruk ja Nøhr (2016) tarjoavat käytettävyysongelmiin ratkaisuksi käyttäjäkeskeisen suunnittelun (engl. user-centered design) ja osallistavan suunnittelun (engl. participatory design). Ne ovat lähestymistapoja, joiden tavoitteena on varmistaa, että järjestelmä vastaa aidosti loppukäyttäjien tarpeita ja tukee heidän työprosessejaan. Kushniruk ja Nøhr (2016) tunnistavatkin loppukäyttäjien, kuten lääkäreiden ja hoitajien, aktiivisen osallistumisen yhdeksi keskeisimmistä menestystekijöistä monimutkaisten terveydenhuollon tietojärjestelmien käyttöönotossa.

Käyttäjien osallistaminen ei tarkoita vain palautteen keräämistä valmiista tuotteesta, vaan iteratiivista prosessia läpi koko kehityskaaren. Tämä voi tapahtua esimerkiksi käyttäjätarvekartoitusten, yhteissuunnittelun (engl. co-design) tai prototyyppien testauksen avulla Dullabh ym. (2023). Alwashmin ym. (2019) mukaan on olennaista, että todelliset käyttäjät pääsevät testaamaan järjestelmää tyypillisissä tehtävissään jo kehitysvaiheessa. Näin käytettävyysongelmat voidaan tunnistaa ja korjata ennen kuin järjestelmä viedään tuotantoon, mikä parantaa sekä järjestelmän laatua että sen hyväksyttävyyttä käyttäjien keskuudessa (Kushniruk & Nøhr, 2016).

## 2.3 Ketterien menetelmien soveltaminen

Fitzgeraldin ym. (2013) mukaan terveydenhuollon ohjelmistokehityksessä menetelmävalintaa ohjaa erityisesti sääntelyyn, turvallisuuteen, jäljitettävyyteen ja laadunvarmistukseen liittyvät seikat. Tämä johtuu siitä, että terveydenhuollon ja lääkinnällisten laitteiden ohjelmistot toimivat ympäristössä, jossa edellytetään laajaa dokumentaatiota, auditoitavuutta sekä prosessien todennettavuutta. Heidän mukaansa nämä vaatimukset korostuvat erityisesti säännellyissä ympäristöissä, joihin terveydenhuollon tietojärjestelmät kuuluvat.

Perinteiset menetelmät, kuten Waterfall- ja V-malli, ovat pitkään olleet standardiratkaisuja terveydenhuollon ohjelmistokehityksessä. Fitzgerald ym. (2013) tunnistavat niiden vahvuudeksi sen, että ne tarjoavat selkeästi määritellyn vaihejärjestyksen, jonka avulla viranomaisille voidaan osoittaa, että kaikki tuotteen määrittelyyn, suunnitteluun, toteutukseen ja testaukseen liittyvät vaatimukset on täytetty. Säännellyissä ympäristöissä painopiste on usein prosesseissa eikä tuotteessa, ja siksi menetelmien ennustettavuus ja dokumentaatio koetaan hyödyllisiksi (Fitzgerald ym., 2013). Kitzmiller

ym. (2006) toisaalta toteavat, että perinteiset menetelmät voivat myös hidastaa kehitystä nopeasti muuttuvassa terveydenhuollon toimintaympäristössä. Heidän mukaansa terveydenhuollon IT-projektit viivästyvät usein, kun suunnittelua painottava lähestymistapa ei vastaa järjestelmien monimutkaisuutta ja muutostarpeita, mikä kasvattaa projektin viivästymis- ja epäonnistumisriskejä.

Ketterien menetelmien käyttöönottoon säännellyssä ympäristössä liittyy monia haasteita, koska ketterien menetelmien periaatteet, kuten dokumentaation minimointi ja muutoksiin reagoiminen, voivat olla ristiriidassa sääntelyvaatimusten kanssa. Fitzgeraldin ym. (2013) QUMAS-tapaustutkimus kuitenkin osoittaa, että ketteriä menetelmiä voidaan käyttää onnistuneesti, jos ne yhdistetään sääntelyn edellyttämään systemaattiseen jäljitettävyyteen ja yksityiskohtaiseen dokumentaatioon. Tapaustutkimuksessa keskeiseksi ratkaisuksi nousi automaattinen jäljitettävyystyökalu, joka yhdisti vaatimukset, testit, koodin ja dokumentaation toisiinsa ja tuotti auditointiin tarvittavat jäljitettävyysmatriisit. Tämä automatisointi mahdollisti sen, että dokumentaatio pysyi ajan tasalla jokaisessa sprintissä ja auditoinneista tuli merkittävästi aiempaa nopeampia ja selkeämpiä. Manjunath ym. (2013) havaitsivat myös, että ketterät menetelmät mahdollistivat riskien aikaisemman havaitsemisen ja paransivat ohjelmiston laatua verrattuna perinteisiin menetelmiin, kunhan järjestelmävaatimuksista huolehdittiin asianmukaisilla työkaluilla.

## 2.4 Terveydenhuollon ohjelmistokehityksen haasteet ja mahdollisuudet

### 2.4.1 Teknisen velan ja monimutkaisen toimintaympäristön haasteet

Terveydenhuollon ohjelmistoissa tärkeimpänä tavoitteena on potilaiden hoidon sujuvoittaminen. Oroviogoicoechean ja Watsonin (2009) mukaan suuri ongelma terveydenhuollon IT-järjestelmien kehityksessä on se, että ohjelmistoja ei suunnitella käytännön tarkoituksiin, vaan vastaamaan lakien edellytyksiin ja hallinnon tarpeisiin. Heidän mukaansa tämä sosiotekninen ristiriita voi heikentää järjestelmän käytettävyyttä ja vaikuttaa negatiivisesti hoitoprosessien sujuvuuteen. Tätä havaintoa tukee Sittigin ym. (2020) näkemys, jonka mukaan terveystietojärjestelmien käyttöönotto tapahtuu monimutkaisissa mukautuvissa järjestelmässä, joissa teknologian, ihmisten ja organisaatiokäytäntöjen välinen vuorovaikutus voi synnyttää uusia turvallisuusriskitekijöitä, joita ei voida ratkaista pelkästään teknisillä korjauksilla.

Kruse ym. (2016) ovat tunnistaneet keskeiseksi ohjelmistokehitystä hidastavaksi tekijäksi terveydenhuollon IT-infrastruktuurin teknisen velan ja vanhentuneet järjestelmät. Heidän mukaansa vanhojen järjestelmien yhteentoimivuusongelmat ja datarakenteiden epäyhtenäisyys ovat suurimpia esteitä uusien teknologioiden hyödyntämiselle. Sittig ym. (2020) tunnistavat myös terveydenhuollon

ohjelmistojen integraatio- ja siirtymäriskit: uusien moduulien, kuten tekoälysovellusten, integroiminen vuosikymmeniä vanhoihin monoliittisiin taustajärjestelmiin on teknisesti vaativaa ja pakottaa kehittäjät käyttämään resursseja ylläpitoon ja purkkaratkaisuihin innovatiivisen kehitystyön sijaan.

Teknisiä haasteita on datan laadussa ja heterogeenisyydessä, mikä vaikeuttaa erityisesti tekoälypohjaisten ratkaisujen kehittämistä. Ghassemin ym. (2021) mukaan terveydenhuollon data on usein epätäydellistä ja pirstaloitunutta. Ohjelmistokehittäjien näkökulmasta tämä tarkoittaa, että roskaa sisään, roskaa ulos -ilmiö (engl. garbage in, garbage out) on jatkuva riski, sillä jos järjestelmän opetusdata on vääristynyttä tai puutteellista, myös generoidun koodin tai algoritmin tuottamat päätökset ovat virheellisiä. Lisäksi Ghassemi ym. (2021) nostavat esiin mallien siirrettävyyden ongelman. Yhdessä sairaalassa toimiva ohjelmisto tai algoritmi saattaa epäonnistua toisessa ympäristössä, koska potilaspopulaatiot ja kirjaamiskäytännöt vaihtelevat. Sittig ym. (2020) tunnistavat saman ongelman potilasturvallisuuden näkökulmasta. He korostavat haastetta varmistaa ohjelmistojen turvallisuus verkottuneissa ja rajapintojen kautta yhdistetyissä kliinisissä ympäristöissä, joissa potilastietojen saumaton jakaminen eri organisaatioiden ja järjestelmien välillä vaatii virheetöntä toimintaa toisiinsa kytketyiltä kokonaisuuksilta.

Erityisesti generatiiviseen tekoälykehitykseen liittyvä haaste on vaatimus läpinäkyvyydestä ja selitettävyydestä (engl. explainability, XAI). Amann ym. (2020) toteavat, että toisin kuin monilla muilla aloilla, terveydenhuollossa ei voida sokeasti luottaa mustan laatikon ratkaisuihin. Sekä lääketieteellinen etiikka että lainsäädäntö edellyttävät, että ohjelmiston tekemät päätökset ovat jäljitettävissä ja perusteltavissa. Tämä asettaa ohjelmistokehitykselle lisävaatimuksen siitä, että koodin tulee olla sekä toimivaa että ymmärrettävää myös ei-teknisille asiantuntijoille, jotta terveydenhuollon ammattilaiset voivat luottaa järjestelmän tuottamaan tietoon. (Amann ym., 2020.) Sittig ym. (2020) korostavat tätä mukaillen, että päätöksenteon tuki täytyy suunnitella turvallisesti siten, että mikäli järjestelmä antaa väärän suosituksen, käyttäjän on pystyttävä arvioimaan sen perusteet virheen välttämiseksi.

Sittig ym. (2020) ovat tunnistaneet yhdeksän lyhyen aikavälin haastetta tietojärjestelmän elinkaaren eri vaiheista alkaen suunnittelusta ja käyttöönotosta valvontaan saakka (Taulukko 1). Taulukossa esitetyt haasteet osoittavat, että Krusen ym. (2016), Amannin ym. (2020) ja Ghassemin ym. (2021) kuvaamien teknisten haasteiden lisäksi tarvitaan myös organisaatiokulttuurin muutosta ja systemaattista osallistamista ja valvontaa.

**Taulukko 1. Sittig ym. (2020) esittämät terveydenhuollon tietojärjestelmien yhdeksän lyhyen aikavälin haastetta**

| Haaste | Kuvaus |
|---|---|
| 1. Riskinarviointimallien ja -menetelmien puute | Ennakoivien, systemaattisten riskinarviointityökalujen puute heikentää mahdollisuutta tunnistaa ja ehkäistä potilasturvallisuutta uhkaavia tilanteita ennen käyttöönottoa. |
| 2. Epäyhtenäiset käyttöliittymät | Eri järjestelmien käyttöliittymästandardien vaihtelu lisää virheiden riskiä ja kasvattaa käyttäjän kognitiivista kuormaa. |
| 3. Turvallisuusriskit järjestelmien välisissä integraatioissa | Kun järjestelmät linkittyvät toisiinsa, syntyy uusia vuorovaikutussuhteita ja haavoittuvuuksia, joita ei aina havaita suunnitteluvaiheessa. |
| 4. Potilaan yksiselitteinen tunnistaminen | Epäselvät tai epäjohdonmukaiset potilastunnisteet voivat johtaa väärien tietojen yhdistymiseen ja vakaviin hoitovirheisiin. |
| 5. Päätöksenteon tuen turvallinen suunnittelu ja integraatio | Päätöksenteon tuki voi parantaa turvallisuutta, mutta huonosti suunniteltuna tai integroituna se voi aiheuttaa uusia riskejä. |
| 6. Järjestelmäsiirtymien turvallinen hallinta | Vanhoista järjestelmistä uusiin siirtyminen voi aiheuttaa tiedonmenetystä, yhteensopivuusongelmia ja virheitä, ellei siirtymää toteuteta kontrolloidusti. |
| 7. Reaaliaikaisen järjestelmämonitoroinnin puute | Monelta organisaatiolta puuttuvat työkalut järjestelmän toiminnan, virheiden ja kuormituksen reaaliaikaiseen seurantaan. |
| 8. Turvallinen raportointi- ja oppimisympäristö (engl. safe harbor) | Organisaatioilla ei ole riittävää oikeudellista tai kulttuurillista tukea virheiden avoimeen raportointiin ja yhteiseen oppimiseen. |
| 9. Potilas- ja kuluttajalähtöisten turvallisuusmallien kehittäminen | Potilaat ja käyttäjät eivät vielä osallistu järjestelmällisesti tietojärjestelmäturvallisuuden kehittämiseen, vaikka heidän näkemyksensä voisivat täydentää ammattilaisten havaintoja. |

### 2.4.2 Standardisaation ja uusien teknologioiden tarjoamat mahdollisuudet

Vaikka terveydenhuollon ohjelmistokehitykseen liittyy merkittäviä haasteita, teknologinen kehitys ja uusien menetelmien omaksuminen tarjoavat myös huomattavia mahdollisuuksia hoidon laadun ja järjestelmien tehokkuuden parantamiseksi. Keskeisimmät mahdollisuudet liittyvät yhteentoimivuuden standardointiin, pilvipalveluihin ja tekoälyavusteiseen kehitykseen.

Merkittävä tekninen mahdollisuus liittyy yhteentoimivuuden standardointiin. Kuten aiemmin todettiin, yhteentoimivuuden puute on ollut pitkäaikainen haaste, mutta HL7 FHIR -standardi (engl. Fast Healthcare Interoperability Resources) tarjoaa ratkaisuja tähän ongelmaan. Saripallen ym. (2019) mukaan FHIR-standardi tarjoaa modernin ja modulaarisen tavan käsitellä terveystietoa, mikä helpottaa potilastietojen siirtämistä eri järjestelmien välillä. Tabari ym. (2024) vievät näkemyksen

pidemmälle, ja korostavat, että FHIR ei ainoastaan ratkaise tiedonsiirron teknisiä ongelmia, vaan se mahdollistaa myös datan laajemman hyödyntämisen tutkimuksessa ja analytiikassa. Heidän mukaansa standardoitu datamalli luo pohjan ylikansalliselle tutkimukselle ja fenotyypitykselle, mikä ei vanhemmilla suljetuilla järjestelmillä ollut mahdollista. Granja ym. (2018) puolestaan tunnistavat yhteentoimivuuden puutteen yhdeksi merkittävimmistä syistä eHealth-interventioiden epäonnistumiseen. Heidän mukaansa standardien noudattaminen ei ole ainoastaan tekninen etu, vaan välttämätön ehto sille, että järjestelmä ylipäätään hyväksytään osaksi kliinistä työnkulkua.

Standardien lisäksi siirtyminen perinteisistä paikallisista järjestelmistä pilvipohjaisiin ratkaisuihin (engl. cloud computing) avaa uusia mahdollisuuksia ohjelmistokehitykselle. Ali ym. (2018) toteavat, että pilvipalvelut tarjoavat terveydenhuollolle skaalautuvuutta ja joustavuutta, jota perinteiset palvelinratkaisut eivät pysty tarjoamaan. Pilvipalvelut mahdollistavat raskaiden laskentatehtävien, kuten lääketieteellisen kuvantamisen analysoinnin ja päätöksenteon tuen, ulkoistamisen kustannustehokkaille alustoille. Tämä teknologinen murros tukee myös yhteentoimivuutta, sillä keskitetyt pilvialustat voivat toimia integraatiopisteinä eri organisaatioiden välillä. (Ali ym., 2018.)

Kolmas keskeinen mahdollisuus liittyy ohjelmistokehityksen menetelmien muutokseen kohti syvempää käyttäjäyhteistyötä. Siinä missä Rytkönen ym. (2020) tunnistavat kehittäjien ja käyttäjien välisen yhteistyön haasteet, he näkevät myös, että onnistunut vuoropuhelu parantaa sovellusten laatua merkittävästi. Dullabh ym. (2023) korostavat, että mahdollisuus ei ole ainoastaan lopputuotteen parempi käytettävyys, vaan koko kehitysprosessin muuttaminen yhteissuunnitteluksi. Kun kliinikot osallistetaan empaattisin menetelmin suunnitteluun, ohjelmistot eivät ainoastaan täytä teknisiä vaatimuksia, vaan voivat aidosti sujuvoittaa hoitopolkuja ja vähentää ammattilaisten kognitiivista kuormaa (Dullabh ym., 2023).

Rodriguezin ym. (2024) tapaustutkimuksessa huomattiin, että ohjelmistokehitysprosessin tehostamiseen generatiivisen tekoälyn avulla liittyy merkittävä mahdollisuus. Heidän mukaansa GenAI-työkalut voivat toimia fasilitaattoreina koko ohjelmistokehityskaaren ajan aina vaatimusmäärittelystä ja käyttäjätarinoiden luomisesta varsinaiseen koodin kirjoittamiseen saakka. Erityisen tärkeä havainto terveydenhuollon kontekstissa oli GenAI:n kyky parantaa teknisten kehittäjien ja ei-teknisten terveydenhuollon ammattilaisten kommunikaatiota. Tekoälyä voidaan siis hyödyntää kääntämään kliiniset tarpeet teknisiksi määrittelyiksi ja toisinpäin, jolloin väärinymmärrysten riski pienenee ja kehitysyhteistyö nopeutuu huomattavasti ilman, että laadusta joudutaan tinkimään. (Rodriguez ym., 2024.)

# 3 Generatiivisella tekoälyllä tuotetun koodin hyödyntäminen terveydenhuollon ohjelmistokehityksessä

## 3.1 Tekoälyavusteiset koodigeneraattorit ja niiden toimintaperiaate

Nykyaikainen tekoälyavusteinen koodin generointi perustuu generatiiviseen tekoälyyn (GenAI) ja erityisesti suuriin kielimalleihin (engl. Large Language Models, LLM). Toisin kuin aiemmat sääntöpohjaiset menetelmät, nämä mallit kykenevät tuottamaan uutta koodia hyödyntämällä syväoppimisen (engl. deep learning, DL) arkkitehtuureja ja suurta määrää opetusaineistoja (Cheng ym., 2025). Odeh ym. (2024) jaottelevat nykyiset tekoälytyökalut kategorioihin, joista merkittävimpiä ovat koneoppimiseen perustuvat mallit (engl. Machine Learning, ML) ja syväoppimismallit. Koneoppimismallit hyödyntävät esimerkiksi tilastollisia kielimalleja oppiakseen säännönmukaisuuksia suurista kooditietokannoista. Odehin ym. (2024) mukaan näitä malleja käytetään tyypillisesti koodin automaattiseen täydentämiseen, mutta ne eivät välttämättä yllä syväoppimismallien tasolle monimutkaisuudessa. Syväoppiminen on laaja käsite, joka kattaa useita erilaisia neuroverkkoarkkitehtuureja, mutta koodin generoinnin kontekstissa merkittävimpiä ovat siihen perustuvat suuret kielimallit. Tällaiset mallit, kuten OpenAI:n Codex ja GPT-4, ovat koulutettu miljardeilla riveillä julkista koodia, mikä mahdollistaa niiden käytön monimutkaisissa tehtävissä, kuten koodin kääntämisessä ohjelmointikielestä toiseen tai luonnollisen kielen komentojen muuttamisessa toimivaksi koodiksi.

Chengin ym. (2025) mukaan GenAI-järjestelmien toiminta rakentuu kolmen toisistaan riippuvaisen ydinkomponentin varaan. Ensimmäinen komponentti muodostuu syvistä neuroverkkoarkkitehtuureista (engl. deep neural architectures), kuten Transformer-mallista, jotka mahdollistavat pitkän aikavälin riippuvuuksien mallintamisen. Toisena keskeisenä tekijänä ovat itseohjautuvat oppimisalgoritmit (engl. self-supervised learning algorithms), joiden avulla mallit oppivat kielen rakenteen ennustamalla peitettyjä tai seuraavia merkkejä ilman erillistä ihmisen tekemää luokittelua. Kolmas komponentti on laajamittainen esikoulutusdata (engl. large-scale pretraining data), joka koostuu valtavista määristä tekstiä ja lähdekoodia, toimien perustana generatiivisten mallien kyvylle jäljitellä ihmismäistä tuotosta.

Vaswanin ym. (2017) esittelemä Transformer-arkkitehtuuri on hallitseva menetelmä kielen mallintamisessa. Sen keskeinen innovaatio on huomiomekanismi (engl. self-attention mechanism), joka mahdollistaa syötteen käsittelyn rinnakkaisesti ja tunnistaa riippuvuussuhteita syötteen eri osien välillä riippumatta niiden etäisyydestä toisistaan. Tämä mekanismi on erityisen kriittinen

ohjelmoinnissa, jossa koodin looginen eheys vaatii usein kaukana toisistaan sijaitsevien elementtien, kuten muuttujien määrittelyn ja niiden käytön, välisten yhteyksien ymmärtämistä (Vaswani ym., 2017).

GenAI generoi koodia ennustamalla todennäköisyyksiä. Ebertin ja Louridaksen (2023) mukaan suuri kielimalli vastaanottaa syötteenä (engl. prompt) luonnollista kieltä tai koodikatkelman, joka muutetaan numeerisiksi vektoreiksi. Tämän jälkeen kielimalli laskee ja ennustaa seuraavan todennäköisimmän sanan tai merkin (engl. token) hyödyntäen edeltävää kontekstia ja mallin oppimia tilastollisia säännönmukaisuuksia (Ebert & Louridas, 2023).

Vaikka kielimallien perusmekanismi perustuu todennäköisyyksiin, generatiivisen tekoälyn kyvykkyyksiä on laajennettu ohjaamalla malleja simuloimaan inhimillistä ongelmanratkaisuprosessia. Wei ym. (2022) osoittivat, että niin kutsuttu ajatusketjupäättely (engl. Chain-of-Thought, CoT) parantaa merkittävästi mallien suorituskykyä monimutkaisissa tehtävissä pakottamalla ne tuottamaan loogisia välivaiheita ennen lopullista vastausta. Uusimmissa päättelymalleissa tämä prosessi on viety pidemmälle, jolloin malli generoi piilossa olevia ajatus-tokeneita (engl. thinking tokens) logiikkansa tarkistamiseksi ja virheiden korjaamiseksi. Vaatimusmäärittelyn kontekstissa Chengin ym. (2025) systemaattinen kirjallisuuskatsaus vahvistaa tämän lähestymistavan hyödyt: CoT-menetelmä parantaa erityisesti vaatimusristiriitojen havaitsemista ja tulosten tulkittavuutta verrattuna perinteisiin menetelmiin. Tästä huolimatta Cheng ym. (2025) havaitsivat, että edistyneiden päättelytekniikoiden käyttö on alalla vielä vähäistä (14,0 %) verrattuna yksinkertaisempiin zero-shot- ja few-shot-tekniikoihin, mikä viittaa siihen, että GenAI-mallien syvällisempää päättelykykyä ei vielä hyödynnetä täysimääräisesti ohjelmistokehitysprosesseissa.

Markkinoilla on tällä hetkellä useita erilaisia koodigeneraattoreita, joiden suorituskyvyissä on eroja. Idrisov ja Schlippe (2024) vertailivat tutkimuksessaan seitsemää eri generatiivista tekoälymallia. Heidän tulostensa mukaan mallien kyky tuottaa toimivaa koodia vaihtelee huomattavasti riippuen käytetystä ohjelmointikielestä ja tehtävän vaikeusasteesta. Mallit eroavat toisistaan taustalla olevan kielimallin, lisenssien ja käyttötarkoitusten osalta. Taulukossa 2 vertaillaan alan keskeisiä tekoälytyökaluja perustuen Ebertin ja Louridaksen (2023), Idrisovin ja Schlippen sekä Wheatonin ja Wheatonin (2025) katsauksiin. Tekoälytyökalujen valinta tähän katsaukseen perustuu niiden näkyvyyteen ja käsittelyyn ajankohtaisessa tutkimuskirjallisuudessa. Taulukkoa on täydennetty uudempien mallien osalta kehittäjien omilla teknisillä raporteilla (Anthropic, 2024) sekä uusimman Lovable-sovelluksen osalta kehittäjän omien nettisivujen (Lovable, 2025) perusteella. Mukana on sekä kaupallisia

suljetun lähdekoodin työkaluja ja malleja, että tutkimuksellisesti merkittäviä avoimen lähdekoodin malleja.

**Taulukko 2. Keskeiset tekoälypohjaiset koodigeneraattorit ja niiden ominaisuudet**

| Työkalu | Kehittäjä | Taustalla oleva teknologia | Lisenssi ja saatavuus | Ominaisuudet ja käyttötapaukset |
|---|---|---|---|---|
| Github Copilot | GitHub / Microsoft | OpenAI Codex (GPT-3-pohjainen) | Kaupallinen (suljettu) | Integroituu suoraan kehitysympäristöön (IDE). Tarjoaa reaaliaikaista koodin täydennystä ja Chat-toiminnon koodin selittämiseen. Suoriutui parhaiten Idrisovin ja Schlippen (2024) vertailussa ratkaisten 50 % testatuista ohjelmointiongelmista. |
| Claude | Anthropic | Claude 3 -malliarkkitehtuuri | Kaupallinen / Freemium | Tunnettu Artifacts-ominaisuudesta, joka mahdollistaa koodin ja käyttöliittymän (UI) reaaliaikaisen esikatselun ja iteratiivisen kehittämisen. Erityisen vahva vaatimusmäärittelyssä ja teknisten dokumenttien luomisessa. |
| Lovable | Lovable | Useat LLM:t, kuten Claude 3.5 ja GPT-4 | Kaupallinen / Freemium | Edustaa vibe-koodauksen lähestymistapaa: generoi kokonaisia full-stack sovelluksia luonnollisen kielen kuvauksista. Mahdollistaa nopean prototypoinnin ilman syvällistä koodausosaamista. |
| Amazon CodeWhisperer | Amazon (AWS) | Proprietary ML Model | Kaupallinen (suljettu) | Erikoistunut AWS-pilvipalveluiden koodiin. Sisältää tietoturvaskannauksia ja huomioi yrityskäytön tietosuojavaatimukset. Idrisovin ja Schlippen (2024) vertailussa ei kyennyt ratkaisemaan yhtäkään annetuista ohjelmointiongelmista. |
| Tabnine | Tabnine | Omat mallit (koulutettu avoimella lähdekoodilla) | Kaupallinen / Freemium | Painottaa yksityisyyttä. Mahdollisuus ajaa paikallisesti omalla koneella, jolloin koodi ei siirry pilveen. Koulutettu ainoastaan lisensoidulla avoimella lähdekoodilla. |
| ChatGPT | OpenAI | GPT-3.5 / GPT-4 | Kaupallinen / Freemium | Yleiskäyttöinen kielimalli. Ei integroidu suoraan editoriin ilman lisäosia, mutta kykenee selittämään koodia, generoimaan testitapauksia ja refaktoroimaan laajasti. Idrisovin ja Schlippen (2024) vertailussa ratkaisi 22,2 % testatuista ohjelmointiongelmista. |

| Työkalu | Kehittäjä | Taustalla oleva teknologia | Lisenssi ja saatavuus | Ominaisuudet ja käyttötapaukset |
|---|---|---|---|---|
| Code Llama | Meta AI | Llama 2 -pohjainen malli | Avoin (Open Weights) | Suorituskykyinen avoin malli, joka tukee pitkiä konteksteja (jopa 100 000 tokenia). Tutkimuskäyttöön ja kaupalliseen käyttöön soveltuva, mahdollistaa mallin hienosäädön. Idrisovin ja Schlippen (2024) vertailussa ratkaisi 22,2 % annetuista ohjelmointiongelmista. |
| StarCoder | BigCode (Hugging Face) | StarCoderBase | Avoin (Open Access) | Koulutettu The Stack -aineistolla, joka sisältää vain sallivasti lisensoitua koodia. Tarjoaa läpinäkyvyyttä siihen, millä datalla malli on opetettu. Idrisovin ja Schlippen (2024) vertailussa ratkaisi 5,6 % annetuista ohjelmointiongelmista. |
| Google Bard (nyk. Gemini) | Google | Gemini Pro / Ultra (aiemmin PaLM 2) | Kaupallinen / Freemium | Kilpailee suoraan GPT-4:n kanssa. Gemini Code Assist integroituu Google Cloudiin ja IDE-ympäristöihin. Tukee erittäin laajaa konteksti-ikkunaa (miljoonaa tokenia), mikä auttaa laajojen kooditietokantojen analysoinnissa |
| Code T5 / InstructCode T5+ | Salesforce Research | Encoder-Decoder (T5-pohjainen) | Avoin (BSD-3-Clause) | Hyödyntää tunnistetietoista esikoulutusta. Vahva koodin ymmärtämisessä, kuten koodin kääntämisessä kielestä toiseen, tiivistämisessä ja virheiden korjaamisessa. Idrisovin ja Schlippen (2024) vertailussa ratkaisi 5,6 % annetuista ohjelmointiongelmista. Tukee 2048 tokenia. |

## 3.2 Generatiivisen tekoälyn hyödyt terveydenhuollon ohjelmistokehitysprosessissa

Generatiivisen tekoälyn hyödyntäminen ohjelmistokehityksessä on ollut erittäin aktiivinen tutkimuskohde viime vuosina, mutta sen soveltaminen terveydenhuollon tiukasti säännellyssä ympäristössä on edelleen nouseva ja kapeampi tutkimusala. Alan ajankohtaisuutta kuvaa joulukuussa 2025 toteutettu haku Scopus-tietokannasta. Hakulauseke, joka edellytti tutkimukselta generatiivisen tekoälyn (esim. ”GenAI”, ”LLM”, ”GPT”), ohjelmistokehityksen (esim. ”software development”, ”code generation”) ja terveydenhuollon kontekstin (esim. ”healthcare”, ”medical”) samanaikaista käsittelyä, tuotti 145 tulosta, joista valtaosa oli viime vuosilta. Tämä suhteellisen suppea tutkimusvolyymi

osoittaa, että vaikka yleinen kiinnostus uusia teknologioita kohtaan on suurta, empiiristä tutkimusta sen turvallisesta integraatiosta juuri terveydenhuollon ohjelmistoprosesseihin on toistaiseksi saatavilla rajallisesti.

Giffarin ym. (2024) mukaan generatiivisen tekoälyn tuomat hyödyt terveydenhuollon ohjelmistokehityksessä kohdistuvat koko ohjelmistokehityksen elinkaareen (engl. Software Development Life Cycle, SDLC) aina vaatimusmäärittelystä ylläpitoon saakka. Odehin ym. (2024) mukaan generatiivisen tekoälyn tuomat hyödyt terveydenhuollon ohjelmistokehityksessä kohdistuvat erityisesti kehityssyklien nopeuttamiseen, kommunikaation parantamiseen teknisten ja kliinisten asiantuntijoiden välillä sekä vanhentuneiden järjestelmien ylläpidon tehostamiseen. Koska terveydenhuollon ohjelmistokehitystä sääntelevät tiukat laatu- ja dokumentaatiovaatimukset, tekoälyn kyky automatisoida rutiinitehtäviä vapauttaa kehittäjien resursseja korkeamman tason ongelmanratkaisuun ja arkkitehtuurisuunnitteluun (Odeh ym., 2024).

Merkittävä prosessihyöty liittyy ohjelmistokehitysprosessin varhaisen vaiheen, eli prototypoinnin ja vaatimusmäärittelyn radikaaliin nopeuttamiseen. Giffari ym. (2024) kuvaavat ohjelmistokehityksen vaatimusmäärittelyvaihetta usein pullonkaulaksi, sillä vaatimukset voivat olla hajallaan eri dokumenteissa, laeissa tai kliinisissä ohjeistuksissa. Tutkimuksessaan he havaitsivat, että analyytikot kokevat GenAI-työkalut merkittäväksi avuksi erityisesti vaatimusten keräämisen yksinkertaistamisessa. Työkalut pystyvät käsittelemään suuria määriä jäsentämätöntä tietoa, kuten PDF-tiedostoja ja kuvia, ja hyödyntämään optista merkintunnistusta (engl. Optical Character Recognition, OCR) tekstin irrottamiseen kehitystyön pohjaksi. Wheaton ja Wheaton (2025) osoittivat tapaustutkimuksessaan, kuinka generatiivinen tekoäly pystyi lyhentämään markkinoilletuloaikaa tuottamalla kattavat vaatimusdokumentit, käyttäjäpolkukuvaukset ja tekniset tarkennukset alle kuudessa tunnissa prosessissa, joka perinteisin menetelmin veisi kuukausia.

Ohjelmistokehityksen yleisessä keskustelussa on noussut esiin käsite vibe-koodaus, jolla Abgrall ym. (2025) viittaavat toimintatapaan, jossa ohjelmistokehitys tapahtuu koodirivien kirjoittamisen sijaan luonnollisella kielellä ohjelmistoa kuvaillen. Tässä lähestymistavassa kehittäjä tai jopa ei-tekninen asiantuntija kuvailee halutun toiminnallisuuden ja tuntuman, jolloin suuret kielimallit tuottavat toimivan koodirungon välittömästi. Vaikka ilmiö ei ole toistaiseksi vakiintunut terveydenhuollon ohjelmistokehityksen käytännöksi, Abgrall ym. (2025) varoittavat jo sen riskeistä: nopeasti ja ilman ymmärrystä generoitu koodi voi olla haurasta ja sisältää tietoturva-aukkoja. Tämä on erityisen kriittistä terveydenhuollon kontekstissa, jossa järjestelmiltä vaaditaan tietoturvan ja tietosuojan kannalta ehdotonta luotettavuutta. Kuviossa 1 vibe-koodaus on sijoitettu vaatimusten määrittelyn ja

ideoinnin vaiheeseen. Tässä vaiheessa menetelmä mahdollistaa nopean kokeilun ja ideoinnin, mutta kuten luvussa 3.4 tarkemmin käsitellään, Abgrallin ym. (2025) mukaan tuotantovaiheessa tällainen koodi vaatii ehdotonta asiantuntijan tarkistusta.

Giffarin ym. (2024) mukaan generatiivisen tekoälyn käyttö mahdollistaa työnjaon muutoksen, sillä GenAI voi tuottaa luonnokset yleisistä tai vähäpätöisemmistä vaatimuksista, jolloin ihmisasiantuntija voi keskittyä syvällisempää teknistä osaamista vaativiin määrittelyihin. Lisäksi se voi nopeuttaa parhaiden käytäntöjen (engl. best practices) löytämistä moduulikohtaiseen kehitykseen, mikä varmistaa, että järjestelmäsuunnittelu pohjautuu ajantasaiseen tietoon ilman aikaa vievää manuaalista taustatyötä.

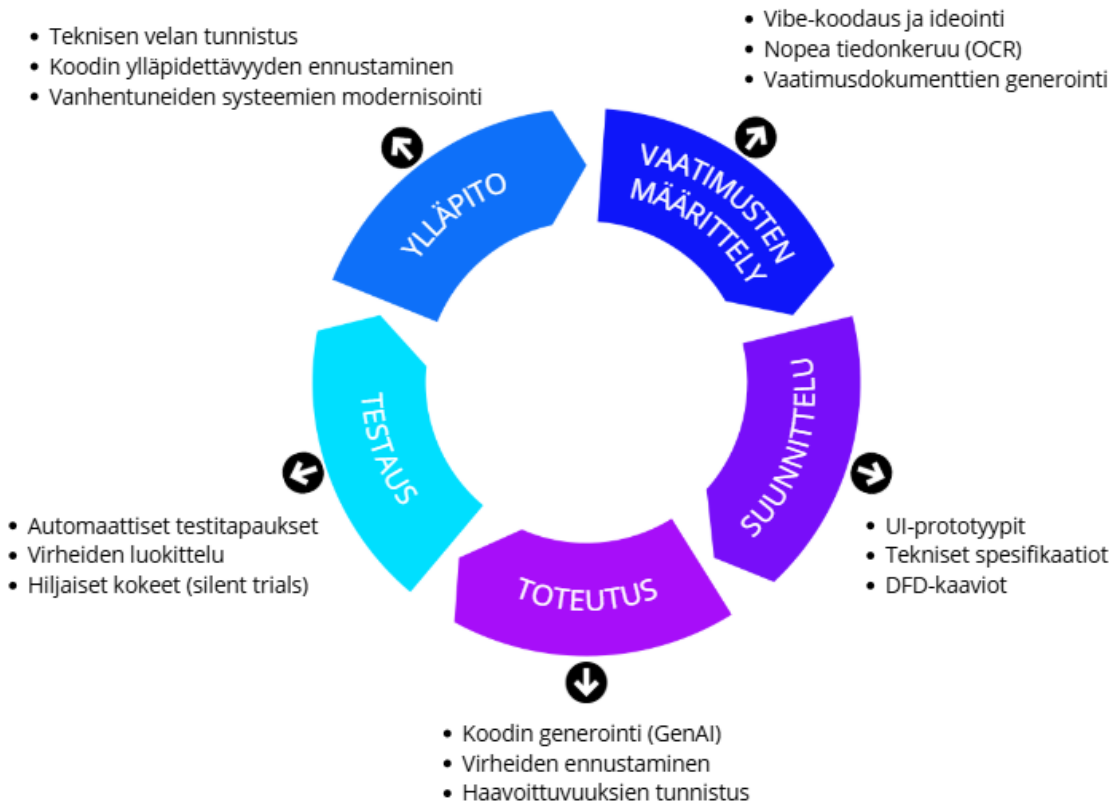
Wheaton ja Wheaton (2025) havaitsivat, että GenAI pystyi selittämään monimutkaisia teknisiä konsepteja ei-teknisille sidosryhmille, mikä paransi tilaajan ja toimittajan välistä yhteisymmärrystä. Generatiivinen tekoäly voi siis vastata kliinisen työn asiantuntijoiden ja ohjelmistokehittäjien välillä olevan kuilun haasteeseen ja toimia tulkkina toimijoiden välillä. Kun GenAI tuottaa välittömästi visuaalisia tai toiminnallisia artefakteja, kuten käyttöliittymäluonnoksia (engl. user interface, UI) keskustelun pohjalta, väärinymmärrysten riski pienenee ja lopputuote vastaa todennäköisemmin käyttäjien todellisia tarpeita. Tämä vähentää kalliita korjauskierroksia projektin myöhemmissä vaiheissa. (Wheaton & Wheaton, 2025.) Giffari ym. (2024) nostavat myös esimerkkinä esiin luonnollisen kielen prosessoinnin (engl. Natural Language Processing, NLP) hyödyntämisen tietovirtakaavioiden (engl. Data Flow Diagrams, DFD) automaattisessa generoinnissa. Tämä tukee Wheatonin ja Wheatonin (2025) havaintoja generatiivisen tekoälyn kyvystä tuottaa visuaalisia artefakteja kommunikaation tueksi.

Odehin ym. (2024) mukaan uuden koodin tuottamisen lisäksi generatiivinen tekoäly tarjoaa merkittäviä hyötyjä olemassa olevien järjestelmien ylläpidossa ja modernisoinnissa. Terveydenhuollon ohjelmistoissa on käytössä paljon vanhentuneita järjestelmiä, joiden ylläpito on kallista ja osaajia on vaikea löytää (Kruse ym., 2016). Odeh ym. (2024) toteavat, että GenAI-mallit kykenevät kääntämään koodia vanhoista ohjelmointikielistä nykyaikaisille kielille sekä refaktoroimaan koodia paremmin ylläpidettäväksi. Giffarin ym. (2024) mukaan sitä voidaan käyttää myös teknisen velan tunnistamiseen ja hallintaan sekä ohjelmiston ylläpidettävyyden ennustamiseen

Lisäksi generatiivisen tekoälyn avulla voidaan automatisoida ohjelmistotestauksen ja dokumentaation luomista. Odehin ym. (2024) mukaan GenAI pystyy analysoimaan laajoja koodikantoja, tunnistamaan kaavoja ja ehdottamaan parannuksia koodin rakenteeseen, mikä parantaa ohjelmiston laatua ja vähentää teknistä velkaa. Automaattinen testitapausten generointi varmistaa, että

järjestelmämuutokset eivät riko olemassa olevia toiminnallisuuksia, mikä on potilasturvallisuuskriittisissä järjestelmissä välttämätöntä. Generatiivinen tekoäly voi tehostaa prosessia automatisoimalla testitapausten luokittelua ja luomista, mikä parantaa testauksen kattavuutta ja nopeuttaa virheiden korjausta. (Odeh ym., 2024.) Lisäksi Sendak ym. (2023) korostavat hiljaisten kokeiden merkitystä. Tekoälyllä voidaan generoida skriptejä, jotka ajavat uutta diagnostiikkamallia rinnakkain tuotantojärjestelmän kanssa keräten dataa sen toimivuudesta ilman potilasriskiä.

Edellä esitettyjen tutkimushavaintojen pohjalta on laadittu yhteenveto generatiivisen tekoälyn hyödyistä terveydenhuollon ohjelmistokehityksen eri vaiheissa. Kuvio 1 on muodostettu syntetisoimalla tässä alaluvussa käsiteltyä aiempaa tutkimuskirjallisuutta ja ryhmittelemällä tunnistetut käyttökohteet ohjelmistokehityksen elinkaaren eri vaiheisiin.




**Kuvio 1 Generatiivisen tekoälyn hyödyt ohjelmistokehityksen eri vaiheissa**

Laadittu mukaillen: Odeh ym. (2024), Giffari ym. (2024), Wheaton & Wheaton (2025), Abgrall ym. (2025), Sendak ym. (2023)

## 3.3 Käyttötapaukset terveydenhuollon kontekstissa

Generatiivinen tekoäly on tuonut ohjelmistokehitykseen uuden toimintamallin, jossa sovelluksia voidaan luonnostella ja jopa toteuttaa käyttämällä luonnollista kieltä ilman perinteistä koodin

kirjoittamista. Menetelmän potentiaalia havainnollistaa Wheatonin ja Wheatonin (2025) tapaustutkimus, jossa hyödynnettiin generatiivista tekoälyä (Claude 3.5 Sonnet) Yhdysvaltain veteraanien terveydenhuoltoon suunnatun Annie Pro -sovelluksen kehityksessä. Prosessissa malli kirjoitti sovelluksen vaatimusmäärittelyt, loi tietokantakaaviot ja generoi käyttöliittymäkoodia pelkkien keskustelunomaisten syötteiden perusteella. Prosessi, joka perinteisesti veisi viikkoja tiivistyi menetelmän avulla alle kuuteen tuntiin. Tällainen lähestymistapa mahdollistaa sen, että terveydenhuollon substanssiosaaja voi ohjata sovelluskehitystä reaaliajassa, mikä vähentää väärinymmärrysten riskiä tilaajan ja toimittajan välillä (Wheaton & Wheaton, 2025).

Diagnostiikkajärjestelmissä koodin virheettömyys on potilasturvallisuuskysymys. GitHub Copilotin ja vastaavien työkalujen yksi vahvimmista sovellusalueista on yksikkö- ja integraatiotestien automaattinen generointi. Odeh ym. (2024) kuvaavat, kuinka GenAI-työkalut pystyvät analysoimaan kirjoitetun koodin logiikan ja ehdottamaan sille kattavia testitapauksia, mukaan lukien harvinaiset reunatapaukset (engl. edge cases), joita ihminen ei välttämättä tulisi ajatelleeksi. Generatiivisen tekoälyn avulla voidaan siis systemaattisesti varmistaa koodin toimivuus myös poikkeavilla syötearvoilla ja virhetilanteissa. Sendak ym. (2023) korostavat tässä hiljaisten kokeiden (engl. silent trials) merkitystä, joissa tekoälytyökalua testataan tuotantodatalla ilman, että se vaikuttaa potilaan hoitoon. Generatiivista tekoälyä voidaan hyödyntää tässä vaiheessa luomaan tarvittavia skriptejä ja testausinfrastruktuuria, jotka ajavat uutta diagnostiikkamallia rinnakkain tuotantojärjestelmän kanssa (Odeh ym., 2024). Näin kokeiden tekninen toteutus nopeutuu ja dataa mallin toimivuudesta saadaan kerättyä turvallisesti.

Terveydenhuollon IT-infrastruktuuri nojaa nykyään tyypillisesti vuosikymmeniä vanhoihin legacy-järjestelmiin, jotka on saatettu kirjoittaa vanhentuneilla kielillä. Näiden järjestelmien ylläpito sitoo resursseja, niiden yhteentoimivuus uusien ratkaisujen kanssa on heikkoa ja integrointi nykyaikaisiin pilvipalveluihin vaikeaa. Odeh ym. (2024) tunnistavat generatiivisen tekoälyn kyvyn tukea ohjelmiston ylläpitoa, kääntää koodia kielestä toiseen ja refaktoroida koodia merkittäviksi sovellusalueiksi. Näistä havainnoista voidaan todeta, että generatiivinen tekoäly tarjoaa ratkaisun Krusen ym. (2016) kuvaamaan ongelmaan: GenAI-työkalu voi analysoida vanhan koodikannan ja auttaa kääntämään sen nykyaikaisille kielille vähentäen manuaalista työtä. Odeh ym. (2024) korostavat myös generatiivisen tekoälyn kykyä oppia olemassa olevasta koodista ja jakaa siihen kätkettyä tietoa, mikä on kriittistä dokumentoimattoman legacy-koodin ymmärtämisessä ja hiljaisen tiedon siirtämisessä.

GenAI:n hyöty sovellusten ydinlogiikan koodaamisessa ilmeni Rodriguezin ym. (2024) tapaustutkimuksessa, jossa he käyttivät generatiivista tekoälyä (GPT-4) luodakseen uudelleen diabeteksen

ehkäisyyn tarkoitetun PAMS-järjestelmän (engl. Personal Automatic Messaging System) ohjelmistokomponentteja. Tutkimuksessa ChatGPT:lle annettiin ohjelmistokehittäjän rooli ja sitä pyydettiin kirjoittamaan Scala-kielellä funktio, joka laskee potilaiden sitoutumisen trendejä kolmen viikon ajanjaksolla. Aluksi tekoälyn tuottama koodi sisälsi loogisia virheitä ja se laski vain viikoittaisia arvoja, mutta iteratiivisella promptauksella ja ihmisasiantuntijan palautteella tekoäly pystyi korjaamaan koodin toimivaksi. Rodriguez ym. (2024) totesivat, että tekoäly toimii tehokkaana fasilitaattorina koodausprosessissa, vaikka monimutkainen logiikka vaatikin kokeneen kehittäjän tarkistusta lopputuloksen ja koodistandardien varmistamiseksi. Tämä havainto korostaa tekoälyn käyttöön liittyvän laadunvarmistuksen tarvetta, jota käsitellään tarkemmin luvussa 3.4.

## 3.4 Tekoälyllä generoidun koodin rajoitukset ja riskitekijät

Luonnolliseen kieleen perustuvan ohjelmoinnin ja generatiivisen tekoälyn odotetaan nopeuttavan ohjelmistokehitystä radikaalisti. Esimerkiksi Wheaton ja Wheaton (2025) raportoivat tapaustutkimuksessaan, että generatiivinen tekoäly lyhensi sovelluskehityksen kuukausista vain kuuteen tuntiin. Tämä nopeus tuo kuitenkin mukanaan merkittäviä laatuun ja turvallisuuteen liittyviä haasteita. Abgrall ym. (2025) varoittavatkin ”nopeuden ilman ymmärrystä olevan valheellista säästöä”. Heidän mukaansa vibe-koodaus voi tuottaa haurasta ja turvatonta ohjelmistoa, sillä jopa puolet tekoälyn generoimasta koodista voi sisältää hyödynnettävissä olevia tietoturvahaavoittuvuuksia, jotka periytyvät mallien opetusdatasta.

Suomen terveydenhuollon säännellyssä toimintaympäristössä tämä aikasäästö onkin todennäköisesti näennäinen. Kun Abgrallin ym. (2025) kuvaamat tietoturvariskit yhdistetään Fitzgeraldin ym. (2013) kuvaamaan vaatimukseen prosessien täydellisestä todennettavuudesta, vibe-koodauksen tuottama nopeushyöty uhkaa huveta jälkitarkastukseen. Koska laki sosiaali- ja terveydenhuollon asiakastietojen sähköisestä käsittelystä (784/2021) vaatii tietojärjestelmiltä ehdotonta tietoturvaa, luottamuksellisuutta ja eheyttä, generatiivisen tekoälyn rooli ei nykyisellään voi olla Suomessa itsenäinen koodaaja terveydenhuollon ohjelmistokehityksessä. Se voi toimia pikemminkin luonnostelijana, jonka tuotokset vaativat perusteellisen ihmisvetoisen auditointiprosessin ennen tuotantokäyttöä.

Teknisten haavoittuvuuksien lisäksi generatiivisen tekoälyn käyttöön liittyy luotettavuusongelmia, jotka johtuvat kielimallien toimintaperiaatteesta. Koska monet GenAI-mallit eivät ymmärrä koodin semantiikkaa vaan ennustavat todennäköisyyksiä, ne voivat tuottaa vakuuttavan näköistä mutta toimimatonta tai virheellistä koodia (Ebert ja Louridas, 2023). Tätä ilmiötä kutsutaan hallusinoinniksi. Idrisovin ja Schlippen (2024) vertailututkimus osoitti, että jopa markkinoiden johtavat mallit

epäonnistuvat monimutkaisissa ohjelmointitehtävissä usein. Heidän testissään parhaiten suoriutunut Github Copilot ratkaisi vain 50 % annetuista ongelmista, kun taas heikoimmat mallit eivät kyenneet ratkaisemaan niistä yhtäkään.

Ebertin ja Louridaksen (2023) mukaan tekoälygeneroidun koodin ymmärrettävyys ja testattavuus voivat tuottaa haasteita. Jos tekoäly tuottaa monimutkaista koodia, jonka toimintalogiikkaa kehittäjä ei täysin ymmärrä, virheiden jäljitettävyys heikkenee. Lisäksi Ghassemi ym. (2021) muistuttavat, että tekoälymallit saattavat oppia virheellisiä korrelaatioita epätäydellisestä datasta, mikä johtaa vääristyneisiin lopputuloksiin.

Tekijänoikeudet ja lisensointi muodostavat oikeudellisen riskin, joka voi altistaa terveydenhuollon organisaatiot juridisille haavoittuvuuksille. Suuret kielimallit on koulutettu miljardeilla riveillä julkista koodia (Odeh, 2024). Suljetun lähdekoodin kaupallisten mallien kohdalla ei useinkaan ole läpinäkyvyyttä siitä, onko opetusdata sisältänyt rajoittavilla lisensseillä suojattua materiaalia. Tällöin ei voida varmistua siitä, että tekoälyn generoima koodi ei riko tekijänoikeuksia. Avoimet mallit, kuten StarCoder pyrkivät ratkaisemaan tätä ongelmaa kertomalla avoimesti opetusdatan alkuperän, mutta niiden suorituskyky ei välttämättä yllä kaupallisten kilpailijoiden tasolle (Idrisov ja Schlippe, 2024).

Riskinä on myös sensitiivisen datan vuotaminen, mikäli kehittäjät syöttävät suojaamatonta koodia tai dataa julkisiin kielimalleihin. Ebert ja Louridas (2023) varoittavat, että suljetun lähdekoodin lataaminen ulkoisiin palveluihin voi johtaa immateriaalioikeuksien ja tietosuojan vaarantumiseen. Terveydenhuollon kontekstissa tällainen tietovuoto olisi erityisen kriittinen, sillä kuten ENISA (2023) raportoi, potilastietojen paljastumisella on vakavia seurauksia potilasturvallisuudelle ja organisaation luotettavuudelle.

Näiden riskien hallitsemiseksi tekoälyavusteisessa ohjelmistokehityksessä on noudatettava ehdotonta human-in-the-loop-periaatetta. Rodriguezin ym. 2024 tutkimus osoitti, että vaikka tekoäly toimii tehokkaana fasilitaattorina, monimutkainen logiikka vaatii aina kokeneen asiantuntijan tarkistusta ja korjausta. Abgrall ym. (2025) korostavatkin, että vastuu koodin toimivuudesta ja turvallisuudesta säilyy aina ihmisellä. Tekoälyn rooli on tukea asiantuntijaa rutiinitehtävissä ja ideoinnissa, ei korvata kriittistä ajattelua tai laadunvarmistusta (Odeh, 2024; Rodriguez ym., 2024).

# 4 Yhteenveto ja johtopäätökset

Tässä kandidaatintutkielmassa tarkasteltiin mahdollisuuksia hyödyntää generatiivista tekoälyä terveydenhuollon ohjelmistokehityksessä. Tutkielman tavoitteena oli selvittää, millaisia erityispiirteitä Suomalaisen terveydenhuollon toimintaympäristö asettaa ohjelmistokehitykselle ja kuinka tekoälyavusteista koodin generointia voidaan soveltaa tällä alalla. Vaikka tekoälyavusteisen koodin generoinnin teknologiat ovat globaaleja, tämän työn analyysi ja sen johtopäätökset nojaavat pitkälti Suomen terveydenhuollon sääntelyyn ja vaatimuksiin. Tämän vuoksi työssä esitetyt havainnot soveltuvat parhaiten Suomen terveydenhuollon ohjelmistokehitykseen.

Kirjallisuuskatsauksen tulokset osoittivat, että Suomessa terveydenhuollon järjestelmien odotetaan täyttävän laissa (784/2021) asetetut edellytykset toiminnallisuudesta, tietoturvasta, tietosuojasta ja yhteentoimivuudesta muiden järjestelmien kanssa. Fitzgeraldin ym. (2013) mukaan tällainen säädelty toimintaympäristö on perinteisesti johtanut raskaiden ja dokumentaatiopainotteisten kehitysmenetelmien käyttöön, sillä ne helpottavat vaatimustenmukaisuuden osoittamisen valvoville viranomaisille. Alan merkittävä haaste on lisäksi Krusen ym. (2016) mukaan vanhentuneet legacy-järjestelmät, jotka sitovat ylläpitoresursseja ja vaikeuttavat uusien innovaatioiden käyttöönottoa.

Vastauksena toiseen tutkimuskysymykseen generatiivisen tekoälyn soveltamisesta ohjelmistokehitysprosessiin havaittiin, että GenAI tarjoaa ratkaisuja nimenomaan näihin haasteisiin. Kirjallisuuden perusteella tekoälyn rooli ei kuitenkaan ole kehittäjän korvaajana, vaan toimia Rodriguezin ym. (2024) kuvaamana fasilitaattorina. Tekoälyavusteinen koodin generointi tehostaa ohjelmistokehityksen elinkaarta sen kaikissa vaiheissa. Se kykenee tehostamaan vaatimusmäärittelyä ja kommunikaatiota toimimalla tulkkina teknisten kehittäjien ja kliinisten asiantuntijoiden välillä. Erityisesti Wheatonin ja Wheatonin (2025) kuvaama nopea prototypointi mahdollistaa sen, että kliinikot voivat osallistua sovelluskehitykseen luonnollisella kielellä. Tämä vähentää väärinymmärryksiä ja varmistaa järjestelmän paremman soveltuvuuden hoitotyöhön.

Kommunikaation parantamisen lisäksi GenAI-työkalut nopeuttavat teknistä toteutusta automatisoimalla rutiinitehtäviä, kuten testitapausten generoimista. Odeh ym. (2024) osoittivat, että tämä parantaa ohjelmistojen robustisuutta, sillä tekoäly kykenee tunnistamaan ja testaamaan myös harvinaisia reunatapauksia, joita ihminen ei välttämättä huomaisi. Lisäksi generatiivinen tekoäly tarjoaa työkaluja teknisen velan purkamiseen analysoimalla vanhaa, heikosti dokumentoitua koodia ja tukemalla sen kääntämisessä nykyaikaisille kielille.

Generatiivisen tekoälyn hyödyntäminen legacy-järjestelmien modernisoinnissa sisältää kuitenkin paradoksin siitä, ratkaistaanko vanha tekninen velka luomalla uusia läpinäkymättömiä ratkaisuja. Jos työkalu kääntää tai refaktoroi koodia, jonka logiikkaa se ei ymmärrä, kuten Ebert ja Louridas (2023) huomauttavat, saatamme päätyä tilanteeseen, jossa järjestelmä on koodikieleltään moderni mutta toiminnaltaan entistäkin vaikeaselkoisempi musta laatikko. Siksi generatiivisen tekoälyn suurin arvo modernisoinnissa ei välttämättä olekaan uuden koodin generoiminen, vaan sen kyky toimia Odehin ym. (2024) mainitsemana hiljaisen tiedon paljastajana, joka auttaa ihmiskehittäjää ymmärtämään dokumentoimatonta järjestelmää ennen muutosten tekemistä.

Tämän tutkielman perusteella voidaan todeta, että generatiivinen tekoäly tarjoaa merkittävän keinon purkaa terveydenhuollon tietojärjestelmien teknistä velkaa ja kuroa umpeen kuilua teknisten kehittäjien ja hoitohenkilökunnan välillä. Tiedämme nyt, että generatiivinen tekoäly pystyy nykyisellään tarjoamaan mahdollisuuksia dokumentaation nopeuttamiseen ja testitapausten luomiseen, mikäli se tapahtuu ihmisen valvomana. Tässä työssä tehty tietokantahaku kuitenkin osoitti, että empiiristä tutkimusta aiheesta on terveydenhuollon kontekstissa toistaiseksi vähän.

Tulevaisuudessa tutkimuksen tulisi keskittyä selvittämään mustan laatikon ongelman ratkaisemista säännellyssä ympäristössä: miten tekoälyn generoima koodi voidaan validoida siten, että se täyttää jäljitettävyysvaatimukset ilman, että ihmisen tekemä tarkistustyö syö saadun hyödyn? Lisäksi olisi tärkeää tutkia, voidaanko Abgrallin ym. (2025) kuvaamaa vibe-koodausta soveltaa turvallisesti terveydenhuollon kontekstissa tulevaisuudessa. Vaikka menetelmä ei ole vielä vakiintunut alalle, sen tarjoama nopeus on houkutteleva, ja jatkotutkimuksen tulisi selvittää ne reunaehdot, joilla luonnollisen kielen avulla tapahtuva kehitys saadaan täyttämään potilasturvallisuuden vaatimukset. Tarvitaan myös pitkän aikavälin seurantatutkimuksia siitä, syntyykö näillä uusilla menetelmillä uudenlaista, vaikeasti havaittavaa teknistä velkaa, joka realisoituu vasta vuosien kuluttua.

Johtopäätöksenä voidaan todeta, että tekoälyavusteisen koodin generointi on terveydenhuollossa potentiaalinen, mutta ei riskitön menetelmä. Sen turvallinen hyödyntäminen edellyttää human-in-the-loop-periaatteen noudattamista, jossa vastuu koodin laadunvarmistuksesta ja lopullisista päätöksistä säilyy ihmisasiantuntijalla. Generatiivinen tekoäly ei korvaa ohjelmistokehittäjää ainakaan nykyisessä vaiheessaan, vaan muuttaa heidän työnkuvaansa. Tulevaisuuden organisaatioiden olisi hyvä panostaa paitsi teknologian hankintaan, myös sellaisten prosessien luomiseen, joissa tekoälyn tuoma nopeus ja ihmisen eettinen sekä ammatillinen harkinta kohtaavat turvallisesti.

## Lähteet

Abgrall, G., Chelly Dagdia, Z., Monnet, X., Zeitouni, K., & Arora, A. (2025). From vibe coding to vibe caring: What clinicians can learn. *The Lancet*, *406*(10512), 1561–1562. https://doi.org/10.1016/S0140-6736(25)01807-0

Ali, O., Shrestha, A., Soar, J., & Wamba, S. F. (2018). Cloud computing-enabled healthcare opportunities, issues, and applications: A systematic review. *International Journal of Information Management*, *43*, 146–158. https://doi.org/10.1016/j.ijinfomgt.2018.07.009

Alwashmi, M. F., Hawboldt, J., Davis, E., & Fetters, M. D. (2019). The Iterative Convergent Design for Mobile Health Usability Testing: Mixed Methods Approach. *JMIR mHealth and uHealth*, *7*(4), e11656. https://doi.org/10.2196/11656

Amann, J., Blasimme, A., Vayena, E., Frey, D., Madai, V. I., & the Precise4Q consortium. (2020). Explainability for artificial intelligence in healthcare: A multidisciplinary perspective. *BMC Medical Informatics and Decision Making*, *20*(1), 310. https://doi.org/10.1186/s12911-020-01332-6

Cheng, H., Husen, J. H., Lu, Y., Racharak, T., Yoshioka, N., Ubayashi, N., & Washizaki, H. (2025). Generative AI for Requirements Engineering: A Systematic Literature Review. *Software: Practice and Experience* (2025): 1–30, https://doi.org/10.1002/spe.70029

Anthropic. (2024). *The Claude 3 Model Family: Opus, Sonnet, Haiku*. Anthropic. Noudettu 10. joulukuuta 2025, osoitteesta https://assets.anthropic.com/m/61e7d27f8c8f5919/original/Claude-3-Model-Card.pdf

*Digitaalisuus sosiaali- ja terveydenhuollon kivijalaksi: Sosiaali- ja terveydenhuollon digitalisaation ja tiedonhallinnan strategia 2023–2035*. (2023). Sosiaali- ja terveysministeriö. https://julkaisut.valtioneuvosto.fi/handle/11111/5687

Dullabh, P., Dungan, R., Raj, M., Catlett, M., Weinberg, S., Jimenez, F., Cope, E., Desai, P., Dobes, A., CDSiC Trust and Patient-Centeredness Workgroup, & Hongsermeier, T. (2023). *Trust & Patient-Centeredness Workgroup: Methods for Involving End-users in PC CDS Co-design*. Agency for Healthcare Research and Quality (US). http://www.ncbi.nlm.nih.gov/books/NBK617025/

Ebert, C., & Louridas, P. (2023). Generative AI for Software Practitioners. *IEEE Software*, *40*(4), 30–38. https://doi.org/10.1109/MS.2023.3265877

*Euroopan parlamentin ja neuvoston asetus (EU) 2016/679, annettu 27 päivänä huhtikuuta 2016 (2016). Euroopan unionin virallinen lehti, L 119, 1–88.* https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=oj:JOL_2016_119_R_TOC

European Union Agency for Cybersecurity (EU body or agency), Theocharidou, M., & Lella, I. (2023). *ENISA threat landscape report: Health sector (January 2021 to March 2023)*. Publications Office of the European Union. https://data.europa.eu/doi/10.2824/163953

Fitzgerald, B., Stol, K.-J., O'Sullivan, R., & O'Brien, D. (2013). Scaling agile methods to regulated environments: An industry case study. *2013 35th International Conference on Software Engineering (ICSE)*, 863–872. https://doi.org/10.1109/ICSE.2013.6606635

Ghassemi, M., Oakden-Rayner, L., & Beam, A. L. (2021). The false hope of current approaches to explainable artificial intelligence in health care. *The Lancet Digital Health*, *3*(11), e745–e750. https://doi.org/10.1016/S2589-7500(21)00208-9

Giffari, R., Ridho, M. M., Sensuse, D. I., Hidayat, D. S., & Purwaningsih, E. H. (2024). Analyst's Perception on the Use of AI-based Tools in the Software Development Life Cycle. *Jurnal Sistem Informasi*, *20*(1), 73–87. https://doi.org/10.21609/jsi.v20i1.1399

Granja, C., Janssen, W., & Johansen, M. A. (2018). Factors Determining the Success and Failure of eHealth Interventions: Systematic Review of the Literature. *Journal of Medical Internet Research*, *20*(5), e10235. https://doi.org/10.2196/10235

Idrisov, B., & Schlippe, T. (2024). Program Code Generation with Generative AIs. *Algorithms*, *17*(2), 62. https://doi.org/10.3390/a17020062

Johnson, C. M., Johnson, T. R., & Zhang, J. (2005). A user-centered framework for redesigning health care interfaces. *Journal of Biomedical Informatics*, *38*(1), 75–87. https://doi.org/10.1016/j.jbi.2004.11.005

Kaipio, J., Lääveri, T., Hyppönen, H., Vainiomäki, S., Reponen, J., Kushniruk, A., Borycki, E., & Vänskä, J. (2017). Usability problems do not heal by themselves: National survey on physicians' experiences with EHRs in Finland. *International Journal of Medical Informatics*, *97*, 266–281. https://doi.org/10.1016/j.ijmedinf.2016.10.010

Kitzmiller, R., Hunt, E., & Sproat, S. (2006). Adopting Best Practices. *CIN: Computers, Informatics, Nursing*, *24*(2), 75–82.

Kruse, C. S., Goswamy, R., Raval, Y. J., & Marawi, S. (2016). Challenges and Opportunities of Big Data in Health Care: A Systematic Review. *JMIR Medical Informatics*, *4*(4), e5359. https://doi.org/10.2196/medinform.5359

Kushniruk, A., Nøhr, C. (2016). Participatory Design, User Involvement and Health IT Evaluation. Teoksessa *Evidence-Based Health Informatics* (s. 139–151). IOS Press. https://doi.org/10.3233/978-1-61499-635-4-139

Laki sosiaali- ja terveydenhuollon asiakastietojen käsittelystä 703/2023. https://www.finlex.fi/fi/lainsaadanto/2023/703?language=fin

Lovable. *Home page*. Noudettu 1. joulukuuta 2025 osoitteesta https://lovable.dev

Manjunath, K. N., Jagadeesh, J., & Yogeesh, M. (2013). Achieving quality product in a long term software product development in healthcare application using Lean and

Agile principles: Software engineering and software development. *Proc. - IEEE Int. Multi Conf. Autom., Comput., Control, Commun. Compressed Sens., iMac4s*, 26–33. https://doi.org/10.1109/iMac4s.2013.6526379

Odeh, A., Odeh, N., & Mohammed, A. S. (2024). A Comparative Review of AI Techniques for Automated Code Generation in Software Development: Advancements, Challenges, and Future Directions. *TEM Journal*, *13*(1), 726–739. https://doi.org/10.18421/TEM131-76

Oroviogoicoechea, C., & Watson, R. (2009). A quantitative analysis of the impact of a computerised information system on nurses' clinical practice using a realistic evaluation framework. *International Journal of Medical Informatics*, *78*(12), 839–849. https://doi.org/10.1016/j.ijmedinf.2009.08.008

Stewart, C. (2025). *Revenue in the healthcare IoT market worldwide* [Tilasto]. Statista. https://www.statista.com/forecasts/1491089/healthcare-iot-revenue-worldwide

Rodriguez, D. V., Lawrence, K., Gonzalez, J., Brandfield-Harvey, B., Xu, L., Tasneem, S., Levine, D. L., & Mann, D. (2024). Leveraging Generative AI Tools to Support the Development of Digital Solutions in Health Care Research: Case Study. *JMIR Human Factors*, *11*(1), e52885. https://doi.org/10.2196/52885

Rytkönen, J., Kinnunen, U.-M., & Martikainen, S. (2022). Sosiaali- ja terveydenhuollon tietojärjestelmäkehittäjien kokemuksia yhteistyöstä käyttäjien kanssa. *Finnish Journal of eHealth and eWelfare*, *14*(2). https://doi.org/10.23996/fjhw.109908

Saripalle, R., Runyan, C., & Russell, M. (2019). Using HL7 FHIR to achieve interoperability in patient health record. *Journal of Biomedical Informatics*, *94*, 103188. https://doi.org/10.1016/j.jbi.2019.103188

Sendak, M., Vidal, D., Trujillo, S., Singh, K., Liu, X., & Balu, S. (2023). Editorial: Surfacing best practices for AI software development and integration in healthcare. *Frontiers in Digital Health*, *5*. https://doi.org/10.3389/fdgth.2023.1150875

Sittig, D. F., Wright, A., Coiera, E., Magrabi, F., Ratwani, R., Bates, D. W., & Singh, H. (2020). Current challenges in health information technology–related patient safety. *Health Informatics Journal*, *26*(1), 181–189. https://doi.org/10.1177/1460458218814893

Valvira. (ei pvm.). *Sosiaali- ja terveydenhuollon tietojärjestelmät*. Noudettu 5. lokakuuta 2025, osoitteesta https://valvira.fi/sosiaali-ja-terveydenhuollon-tietojarjestelmat

Tabari, P., Costagliola, G., Rosa, M. D., & Boeker, M. (2024). State-of-the-Art Fast Healthcare Interoperability Resources (FHIR)–Based Data Model and Structure Implementations: Systematic Scoping Review. *JMIR Medical Informatics*, *12*, e58445. https://doi.org/10.2196/58445

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6000–6010.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems, 35, 24824–24837.*

Wheaton, K. J., & Wheaton, C. F. (2025). From Code to Care: AI and Human Collaboration in Crafting Veteran-Centric Health Technology. *Military Medicine*, *190*(9–10), e1882–e1888. https://doi.org/10.1093/milmed/usaf163

Yen, P.-Y., & Bakken, S. (2012). Review of health information technology usability study methodologies. *Journal of the American Medical Informatics Association*, *19*(3), 413–422. https://doi.org/10.1136/amiajnl-2010-000020

# 5 Liitteet

## Liite 1 Selvitys tekoälyn käytöstä

Tämän tutkielman valmistelussa on hyödynnetty tekoälypohjaisia työkaluja tukemaan tutkimus- ja kirjoitusprosessin eri vaiheita. Tekoälyä on käytetty seuraavilla osa-alueilla:

Google Geminiä hyödynnettiin tutkielman rakenteen ideoinnissa. Tekoälyn antamaa palautetta käytettiin sen varmistamiseksi, että luvut etenevät loogisessa järjestyksessä ja tutkimuskysymyksiin vastataan johdonmukaisesti. Tekoälyä käytettiin myös tukena lukujen otsikoinnin ideoinnissa, mutta lopullinen rakenne ja otsikointi ovat kirjoittajan laatimia.

Lähdekirjallisuuden etsinnässä hyödynnettiin Scopus AI -hakutyökalua. Sitä käytettiin kartoittamaan aihepiirin keskeisintä ja ajankohtaisinta tutkimuskirjallisuutta sekä muodostamaan hakulausekkeita tietokantahakuja varten.

Aineiston hallinnassa käytettiin Google NotebookLM -työkalua. Työkalun avulla luotiin tiivistelmiä PDF-muotoisista tutkimusartikkeleista keskeisten argumenttien ja tulosten nopeaa hahmottamista varten. Tutkielmassa käytetyt lähteet on kuitenkin luettu ja tarkistettu tietojen oikeellisuuden ja kontekstin varmistamiseksi ennen niihin viittaamista.

Viimeisenä Googe Gemini-työkalua käytettiin kielenhuoltoon kieliopin oikeellisuuden varmistamisen tueksi. Kaikki tekoälyn tekemät korjausehdotukset tarkistettiin ja muokattiin sopimaan kirjoittajan omaan ilmaisuun ennen niiden sisällyttämistä lopulliseen työhön.

Kaikissa tapauksissa lopulliset päätökset sisällöstä, tulkinnoista ja tekstin muotoilusta on tehnyt kirjoittaja. Tekoäly on toiminut ainoastaan avustavassa roolissa, eikä se ole tuottanut itsenäisesti lopullista tekstiä tai tehnyt tieteellisiä johtopäätöksiä. Tutkielman toteutuksessa on noudatettu Turun yliopiston linjausta tekoälyn käytöstä.