

Omar Harb

GENERATIIVISEN TEKOÄLYN HYÖDYNTÄMINEN OHJELMISTOTESTAUKSESSA

Kandidaatintutkielma
Johtamisen ja talouden tiedekunta
Tarkastaja: Henri Jalo
Joulukuu 2023

TIIVISTELMÄ

Omar Harb: Generatiivisen tekoälyn hyödyntäminen ohjelmistotestauksessa
Utilizing generative artificial intelligence in software testing
Kandidaatintutkielma
Tampereen yliopisto
Johtamisen ja talouden tiedekunta
Teknis-taloudellinen kandidaattiohjelma
Joulukuu 2023

Tekoälyn ja etenkin generatiivisen tekoälyn räjähdysmäinen kasvu viime vuosina on vaikuttanut merkittävästi eri toimialoihin ja ammatteihin. Generatiivinen tekoäly on itsessään uusi konsepti ja tekoälyosaamista on saatavilla vain rajattu määrä, jonka takia aiheeseen liittyvää kirjallisuutta on vain rajallinen määrä. Ohjelmistotestauksesta taas löytyy laajasti lähdemateriaalia. Tekoälyn käyttö on ollut monessa organisaatiossa ajankohtainen asia, sillä sen avulla on mahdollista saavuttaa kilpailuetua.

Tämän kandidaatintyön tarkoituksena on selvittää, miten generatiivista tekoälyä hyödynnetään ohjelmistotestauksessa. Tutkimus on toteutettu systemaattisena kirjallisuuskatsauksena. Lisäksi työssä on hyödynnetty helmenkasvatusmenetelmää lähdeaineiston keräämisessä. Tutkimusaineisto koostuu tieteellisistä artikkeleista sekä konferenssijulkaisuista. Tutkimus on jaettu kahteen osaa. Ensimmäisessä osassa esitetään määritelmä tekoälylle sekä ohjelmistotestaukselle ja testausprosessille. Toisessa osassa hyödynnetään testausprosessin sekä tekoälyn määritelmää ja esitellään tekoälyn mahdollisuuksia ohjelmistotestauksessa. Toisen osan lopussa esitellään työn tulokset eli vastaukset tutkimuskysymyksiin ja niiden perusteella tehtyjä päätelmiä kuten jatkotutkimusehdotukset.

Tutkimuksessa käytetyn lähdeaineiston tarkastelun avulla tunnistettiin eri keinoja, miten tekoälyä voi hyödyntää testausprosessin eri vaiheissa sekä manuaalisessa että automaattisessa testaamisessa. Tutkimuksen tuloksista voidaan päätellä generatiivisen tekoälyn hyödyntämisen olevan mahdollista, mutta täyttä automaatiota ei ole vielä mahdollista saavuttaa. Keskeisimmiksi haasteiksi tutkimuksessa tunnistettiin vertaisarvioidun kirjallisuuden puute sekä yleisesti aiheeseen liittyvän tutkimuksen määrä. Keskeisimmistä haasteista voidaan päätellä, että jatkotutkimusta kannattaa tehdä vasta muutaman vuoden kuluttua, kun vertaisarvioituja lähteitä on saatavilla enemmän.

Avainsanat: Suuret kielimallit, generatiivinen tekoäly, ohjelmistot, koneoppiminen, ohjelmistotestaus

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

ALKUSANAT

Tämä kandidaatintyö on tehty Tampereen yliopiston tietojohdamisen koulutusohjelman kandidaatintyön kurssilla syksyllä 2023. Työssä tutkitaan generatiivisen tekoälyn käyttöä ohjelmistotestauksessa. Aihe kiinnosti itseäni, koska olen suorittanut useita koneoppimiseen liittyviä kursseja sekä olen töissä ohjelmistotestaajana.

Haluaisin kiittää hyviä ystäviäni Tomi Ylimäkeä, Risto Miettistä, Jan Hautalaa ja Tino Lainetta, jotka tukivat työni valmistumista motivoimalla minua.

Tampereella, 2.11.2023

Omar Harb

SISÄLLYSLUETTELO

1. JOHDANTO	1
1.1 Tutkimuksen tausta	1
1.2 Tutkimuskysymys ja tutkimuksen tavoitteet	2
1.3 Tutkimuksen rakenne	3
2. TUTKIMUSMENETELMÄT JA AINEISTO	4
2.1 Tutkimusmenetelmä	4
2.2 Tutkimusaineisto	5
3. TEKOÄLY	7
3.1 Luonnollisen kielen prosessointi	8
3.2 Suuret kielimallit ja generatiivinen tekoäly	9
4. OHJELMISTOTESTAUS	10
4.1 Manuaalinen testaus	12
4.2 Automaattinen testaus	13
5. GENERATIIVINEN TEKOÄLY JA OHJELMISTOTESTAUKSEN VAIHEET	16
5.1 Uusien testitapausten luominen	16
5.2 Testien ajaminen	17
5.3 Testien sisään- ja ulostulo sekä niiden arviointi	17
5.4 Testitulosten raportointi ja vianmääritys	18
6. YHTEENVETO	19
6.1 Tutkimuksen tulokset	19
6.2 Jatkotutkimusehdotukset	21
LÄHTEET	22

1. JOHDANTO

Tässä luvussa esitellään teoreettinen tausta tutkimukselle ja tarkastellaan tutkimusaihetta sekä sen merkitystä. Tämän jälkeen esitetään tutkimusongelma pää- ja sivututkimuskysymyksen muodossa sekä aiheen rajaukset. Lopuksi esitellään tutkimuksen rakenne.

1.1 Tutkimuksen tausta

Tekoäly (engl. *artificial intelligence*) on järjestelmä, joka suorittaa ihmisen älykkyyteen liittyviä toimintoja kuten päättely, oppiminen ja itsensä kehittäminen (ISO/IEC, 1995). Tekoälyn räjähdysmäinen kasvu viime vuosien aikana on vaikuttanut merkittävästi eri toimialoihin ja ammatteihin (Brandoni, 2023). Organisaatiot, jotka eivät hyödynnä tekoälyä, ovat merkittävästi huonommassa asemassa tekoälyä hyödyntäviin organisaatioihin verrattuna, minkä seurauksena tekoälyn hyödyntämisestä on tullut strateginen välttämättömyys (Lee et al., 2019). Organisaatiot voivat hyödyntää tekoälyä integroimalla sen sovelluksia omiin tuotteisiinsa tai käyttämällä tekoälyä esimerkiksi rekrytoinnissa tai päätöksenteossa.

Ohjelmisto (engl. *software*) on kokonaisuus tietokoneelle tehtyjä toisiinsa liittyviä ohjelmia, dokumentaatiota ja toimintamalleja, jotka vaativat kehittämistä ja muutoksia (IEEE, 1990; Osterweil, 2008). Nykyajan dynaamisessa toimintaympäristössä on elintärkeää tavoitella mahdollisimman korkealaatuista tuotetta tai palvelua. Myös ketterien menetelmien hyödyntäminen on kasvanut teknologian kehityksen myötä. (Hooda, 2023) Ketterät menetelmät tarkoittavat ohjelmistokehityksen näkökulmasta ohjelmistojen nopeaa kehitystä, jossa luodaan asiakkaan tarpeisiin vastaavaa luotettavaa ohjelmistoa (Zannier et al., 2004). Ketterässä ohjelmistokehityksessä on tärkeää, että testauksen yhteydessä ei tuhlata aikaa, sillä testausta tapahtuu jatkuvasti sitä mukaan, kun ominaisuudet valmistuvat. Testauksessa tulee keskittyä oikeiden asioiden tekemiseen sen sijaan että tekee asioita oikein, jolloin testaukseen varattu aika käytetään kustannustehokkaasti (Niittyviita, 2023).

Ohjelmistotestauksen avulla voidaan varmistaa useita asioita kuten tuotteen kestävyys kyberhyökkäyksiä vastaan, lakien ja säädösten noudattaminen, suorituskyky sekä eri

vaatimusten täytyminen. Näistä eri osa-alueista voidaan rakentaa mittareita päätöksenteon tueksi. (Schulmeyer, 2007)

Ohjelmistojen laadunvarmistus (engl. *software quality assurance*) tarkoittaa eri aktiviteetteja, joilla mitataan ja kerätään varmuus siitä, että ohjelmisto soveltuu sille tarkoitettuun käyttöön (IEEE 730, 2014). Laadunvarmistuksessa on siis tarkoitus mitata ohjelmiston laatua ja käsitellä tätä dataa. Tietokoneet pystyvät käsittelemään dataa huomattavasti paremmin ja tarkemmin kuin ihmiset. Tekoälyn avulla pystytään käsittelemään paljon kompleksisempaa dataa, mutta tämä ei silti tarkoita sitä, että ihmiset eivät olisi osana laadunvarmistuksen prosessia. (Goericke, 2020)

1.2 Tutkimuskysymys ja tutkimuksen tavoitteet

Tässä kandidaatintutkielmassa tarkastellaan generatiivista tekoälyä sekä miten organisaatiot hyödyntävät sitä ohjelmistojen laadunvarmistuksessa ja erityisesti testaamisessa. Tavoitteena on siis selvittää generatiivisen tekoälyn hyödyntämisen nykytilanne sekä pohtia lähitulevaisuuden mahdollisuuksia ohjelmistotestaamisen osa-alueissa, joissa tekoälyä ei vielä hyödynnetä. Tutkimuksen tavoitteista määritettiin seuraavanlainen päätutkimuskysymys:

- Miten generatiivista tekoälyä hyödynnetään ohjelmistotestauksessa?

Päätutkimuskysymys voidaan jakaa muutamaaan tutkimusta tukevaan alatutkimuskysymykseen, jotka rajaavat tarkempaa tutkimusta. Päätutkimuskysymyksestä muodostettiin seuraavat alatutkimuskysymykset:

- Miten generatiivista tekoälyä hyödynnetään ohjelmistotestauksen manuaalisessa testaamisessa?
- Miten generatiivista tekoälyä hyödynnetään ohjelmistotestauksen automaattisessa testaamisessa?

Alatutkimuskysymysten avulla saadaan teknisempi näkökulma tutkimukseen, jolloin sen tuloksia voidaan lähestyä useammasta eri näkökulmasta. Ohjelmistotestausta tullaan siis käsittelemään prosessina eikä ohjelmointiin perehdytä esittelyä tarkemmin. Generatiivista tekoälyä käsitellään siten, että mallien toimintaa kuvaillaan ilman että perehdytään matemaattisiin yksityiskohtiin. Myöskään organisaation talouteen liittyviin asioihin ei tulla syventymään, vaikka tekoälyyn liittyy myös paljon taloudellisia ongelmia.

1.3 Tutkimuksen rakenne

Tämä kandidaatintutkielma koostuu kuudesta eri luvusta. Ensimmäisessä luvussa on tarkoituksena esitellä tutkimuksen teoreettista taustaa, tavoitteita rakennetta ja menetelmiä. Toinen luku esittelee tutkimusmenetelmän sekä tutkimusaineiston. Kolmas luku käsittelee generatiivista tekoälyä yleisesti. Luvun tarkoituksena on esitellä suuria kielimalleja, generatiivista tekoälyä sekä niiden toimintaa. Neljäs luku käsittelee ohjelmistotestaamisen osa-alueita ja ohjelmistotestausta yleisesti. Luvut 3 ja 4 koostavat siis tämän kandidaatintutkielman teoriaosuuden. Viidennessä luvussa tutkitaan, miten paljon automaattisessa ja manuaalisessa testaamisessa hyödynnetään generatiivista tekoälyä. Kuudennessa luvussa kootaan tutkimuksen tulokset yhteen, esitetään vastaus päätutkimuskysymykseen sekä pohditaan tulevaisuuden mahdollisuuksia ja mahdollisia jatkotutkimusaiheita.

2. TUTKIMUSMENETELMÄT JA AINEISTO

Tässä luvussa esitellään tutkimuksessa käytetty tutkimusmenetelmä ja syyt sen valintaan. Toiseksi käsitellään tutkimusainestoa ja sen keruuta. Lopuksi käsitellään tutkimusaineisto rajauksia.

2.1 Tutkimusmenetelmä

Tämän kandidaatintutkielma toteutettiin kirjallisuuskatsauksena hyödyntäen Finkin mallia. Kirjallisuuskatsauksen tarkoituksena on selvittää, millaista tietoa aiheesta on saatavilla ja sen vahvuuksia ovat systemaattisuus, toistettavuus sekä täsmällisyys (Fink, 2014). Finkin malli koostuu seitsemästä vaiheesta, jotka ovat:

1. Tutkimuskysymyksen asettaminen
2. Kirjallisuuden ja tietokantojen valitseminen
3. Hakutermien valitseminen
4. Haku/rajauskriteerien valitseminen
5. Metodologinen rajaaminen
6. Kirjallisuuskatsauksen tekeminen
7. Tulosten syntetisointi (Fink, 2014)

Finkin mallin lisäksi tiedonhaussa hyödynnettiin helmenkasvatusmenetelmää, jossa uutta lähdemateriaalia etsitään aiheeseen soveltuvan julkaisun lähdeluettelosta. Tämä mahdollisti muiden aiheeseen liittyvien tai osittain aiheeseen liittyvien kandidaatintutkielmien tai diplomitöiden tutkimisen ja niiden hyödyntämisen tutkimuksessa.

Ennen varsinaisen tutkimuksen aloittamista tehtiin kuitenkin aiheanalyysi, jossa perehdyttiin erilaisiin lähestymistapoihin aiheeseen. Analyysissä käsiteltiin myös alustavasti tutkimusongelmaa, tutkimuksen rajausta, tutkimusaineistoa, tutkimukselle keskeisiä käsitteitä sekä mahdollisia tuloksia. Aiheanalyysin jälkeen tehtiin tutkimussuunnitelma, jossa määritettiin tutkimuksen tutkimusongelma, tutkimuskysymykset, tavoitteet, rakenne sekä perehdyttiin eri tutkimusmenetelmiin ja etsittiin tutkimuksen aineisto.

2.2 Tutkimusaineisto

Tässä kandidaatintutkielmassa tutkimusaineisto valikoitiin pääosin valituista tietokannoista sekä muutama työtehtävien kautta ilmenneistä diplomitoista. Pääasialliseksi tietokannaksi valittiin Tampereen yliopiston palvelu Andor sekä Googlen Scholar. Tutkimusaiheen ajankohtaisuus vaikeutti siihen liittyvien tieteellisten artikkeleiden löytämistä, sillä aihetta ei vielä olla tutkittu paljon.

Lähdemateriaalia oli hankala rajata, sillä hakutuloksia oli suhteellisen vähän. Tutkimuksessa kuitenkin pyrittiin käyttämään mahdollisimman uutta aineistoa. Tutkimuksen laadun kannalta oli tärkeää löytää ajankohtaisia sekä luotettavia lähteitä. Taulukossa 1 on esitetty Scholariin ja Andoriin syötettyjä tutkimuksen keskeisistä käsitteistä muodostettuja hakulausekkeita ja niiden tuottamien tulosten määriä. Teoriaosuudessa lähdeaineistoa rajattiin valitsemalla vuoden 2015 jälkeen tehtyjä artikkeleita. Itse tutkimusosuudessa lähdeaineisto rajattiin tälle vuosikymmenelle eli vuoden 2020 jälkeen tai aikana tehtyihin artikkeleihin ja konferenssijulkaisuihin. Löydettyä lähdeaineistoa rajattiin vielä lisää tarkastelemalla lähteen tiivistelmää, jonka perusteella määritettiin lähteen soveltuvuus luvussa 4.2 esitellyn testausprosessiin. Näiden rajausten avulla lähdeaineistoksi valittiin pääosin vuonna 2023 julkaistuja artikkeleita ja konferenssijulkaisuja.

Taulukko 1: Hakulausekkeet sekä niiden tuottamien tulosten määrät

Hakulauseke	Scholar	Andor
"software testing" AND ("large language models" OR "LLM")	811	23
("generative artificial intelligence" OR "generative AI") AND ("software testing" OR "quality assurance")	687	37
("generative AI" OR "generative artificial intelligence") AND "test case"	246	71
("generative AI" OR "generative artificial intelligence") AND "software testing"	205	11
("generative AI" OR "generative artificial intelligence") AND "manual testing"	38	3
("generative AI" OR "generative artificial intelligence") AND "test automation"	30	4

Kuten taulukosta 1 voidaan nähdä, hakutuloksia ei ollut suurta määrää ja testiautomaation tapauksessa artikkeleita ei löytynyt lähes yhtään. Testiautomaation tapauksessa suurin osa Andorista löytyneistä lähteistä oli ranskaksi, joten niiden hyödyntäminen osoittautui vaikeaksi.

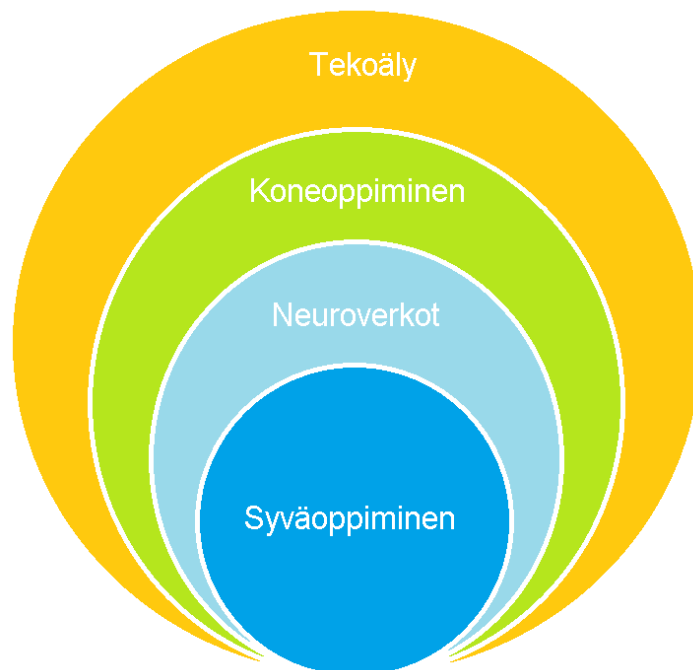
Tässä tutkimuksessa käytettiin lähteitä pääosin tietojenkäsittelytieteen ja tietotekniikan alalta. Tutkimus perustui vahvasti teoriaan, jonka takia teoreettista lähdeaineistoa hyödynnettiin pääosin. Tutkimuksessa käytettävä lähdeaineisto löytyi pääosin hakulausekkeiden avulla, mutta muutamia lähteitä löytyi myös työpaikan sekä yliopiston kurssien kautta. Myös joitain tarkennettuja taulukosta 1 poikkeavia hakulausekkeitä käytettiin lähdeaineiston hankintaan.

3. TEKOÄLY

Tässä luvussa käsitellään tekoälyä yleisellä tasolla sekä perehdytään tarkemmin luonnollisen kielen prosessointiin (engl. *natural language processing*), suuriin kielimalleihin (engl. *large language models*) sekä generatiiviseen tekoälyyn (engl. *generative artificial intelligence*). Luvun tarkoituksena on antaa pintapuolinen kuva tekoälystä ja suurien kielimallien toiminnasta.

Kuten luvussa 1.1 on esitetty, tekoäly on järjestelmä, joka suorittaa ihmisen älykkyyteen liittyviä toimintoja kuten päättely, oppiminen ja itsensä kehittäminen. Aikaisemmin tekoälyä on kuitenkin kuvailtu ohjelmana, joka ei suoriudu ihmistä huonommin satunnaisessa maailmassa (Dobrev, 2012). Tekoäly itsessään on moniselitteinen käsite, joten sen tarkka määrittely on lähes mahdotonta. Moniselitteisyyden takia tekoälystä ei ole vielä tehty standardisoitua määritelmää (Samoili et al., 2020).

Tekoäly on erittäin laaja käsite ja sen voi luokitella pääkäsitteeksi, joka voidaan jakaa seuraaviin alakäsitteisiin: koneoppiminen (engl. *machine learning*), neuroverkot (engl. *neural networks*) ja syväoppiminen (engl. *deep learning*) (Khanagar et al., 2021). Käsitteiden kokonaisuus on visualisoituna kuvassa 1.



Kuva 1: Tekoäly ja sen osajoukot (Khanagar et al., 2021) soveltaen.

Nämä käsitteet ovat siis tekoälyn osajoukkoja. Vaikka nämä käsitteet eivät kuulu tutkimuksen laajuuteen, ovat ne silti tarpeellista määritellä lyhyesti. Määritelmät auttavat ymmärtämään tekoälyä ja tutkimuksen aihetta ja syitä paremmin.

Koneoppiminen tarkoittaa matemaattisia menetelmiä, joilla koneet oppivat erilaisia konsepteja kokemuksen avulla (Hu & Xing, 2022). Khanagar et al. (2022) määrittelee koneoppimisen tekoälyn osajoukkona, joka ennustaa tuloksia tietoaisteista erilaisten algoritmien avulla.

Neuroverkot ovat koneoppimisen osajoukko. Ne ovat kokoelma algoritmeja, jotka luovat signaaleja keinotekoisien neuronien avulla. Neuroverkkojen ideana on simuloida ihmisen aivojen toimintaa. (Khanagar et al., 2021)

Syväoppiminen on taas koneoppimisen ja neuroverkkojen osajoukko, joka käyttää verkkojen useita laskentakerroksia syötetyn datan analysoimiseen. Syväoppiminen tunnetaan myös nimellä konvoluutioneuroverkko. (Khanagar et al., 2021)

3.1 Luonnollisen kielen prosessointi

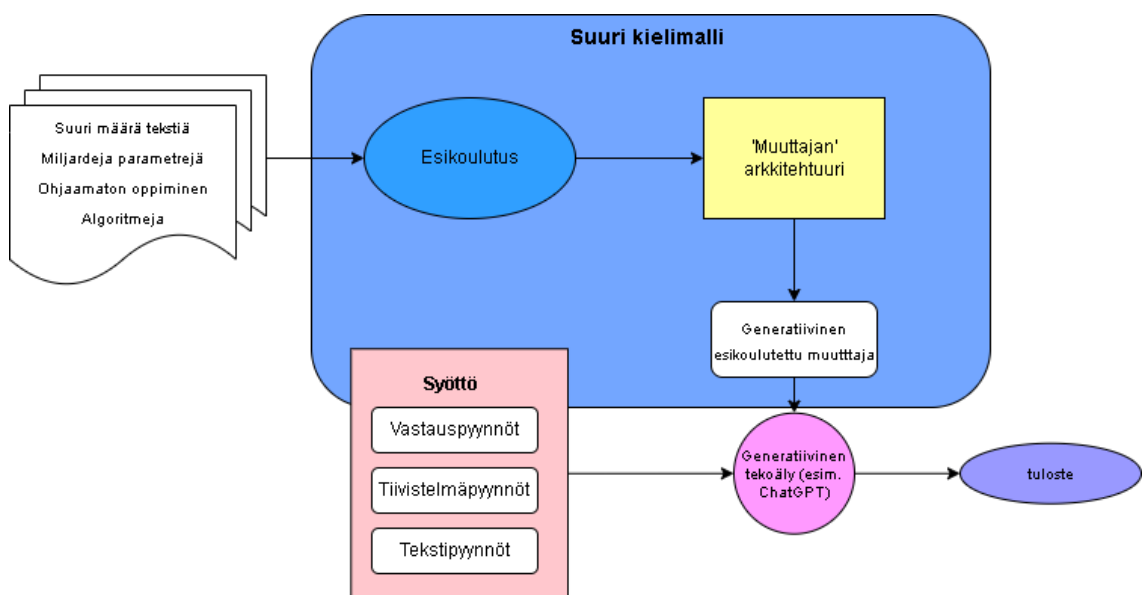
Luonnollisen kielen prosessointi on tekoälyn ja lingvistiikan eli kielitieteen yhdistelmä. Alun perin luonnollisen kielen prosessointi erosi merkittävästi tekstihausta, jossa hyödynnetään skaalautuvia tilastopohjaisia tekniikoita, joilla haetaan suuria määriä tekstiä tehokkaasti. Nykyään nämä ovat kuitenkin lähentyneet paljon. (Nadkarni et al., 2011) Ensimmäisiä käyttökohteita luonnollisen kielen prosessointiin oli yksinkertaiset konekääntäjät kielestä toiseen. Näissä kuitenkin ilmeni ongelmia moniselitteisten sanojen ja sanontojen takia. (Nadkarni et al., 2011)

Kuvassa 1 luonnollisen kielen prosessointi olisi sijoitettu tekoälyn sisälle siten, että se sisältäisi myös koneoppimista, neuroverkkoja sekä syväoppimista. Kang et al. (2020) mukaan luonnollisen kielen prosessointi on tietokoneavusteinen analyttinen tekniikka, joka pyrkii automaattisesti analysoimaan ja ymmärtämään ihmisten kieltä. Luonnollisen kielen prosessointi koostuu kahdesta osa-alueesta: luonnollisen kielen ymmärtämisestä ja luonnollisen kielen generoinnista (Kang et al., 2020). Luonnollisen kielen ymmärtämisen päätehtävänä on ymmärtää luonnollista kieltä tulkitsemalla asiakirjoja ja keräämällä niistä arvokasta tietoa jatkotehtäviä varten. Luonnollisen kielen generointi on taas tekstin luomista luonnollisilla ihmisten ymmärtämillä kielillä. (Kang et al., 2020)

3.2 Suuret kielimallit ja generatiivinen tekoäly

Suuret kielimallit ovat syväoppimismalleja eli konvoluutioneuroverkkoja. Näiden mallien tehtävänä tunnistaa, tiivistää, kääntää, ennustaa ja generoida tekstipohjaista materiaalia erittäin suurista tietoaaineistoista. Nämä mallit ovat koulutettu erittäin suurilla datamäärillä käyttäen ohjaamatonta oppimista (engl. *unsupervised learning*). (Mohapatra et al., 2023)

Generatiivinen tekoäly käyttää suurta kokoelmaa tekstikorpuksia eli tiettyä tarkoitusta varten koottu tekstikokoelma, luodakseen uusia verisoita tekstistä, kuvia tai ennustettua dataa käyttäjän pyynnöstä (Euchner, 2023). Keskitytään tässä tutkimuksessa pääosin tekstiä generoivaan tekoölyyn, jotka ovat toteutettu suurien kielimallien ympärille. Kuvasta 2 nähdään generatiivisen tekoölyn, joka hyödyntää suurta kielimallia, kaaviokuva.



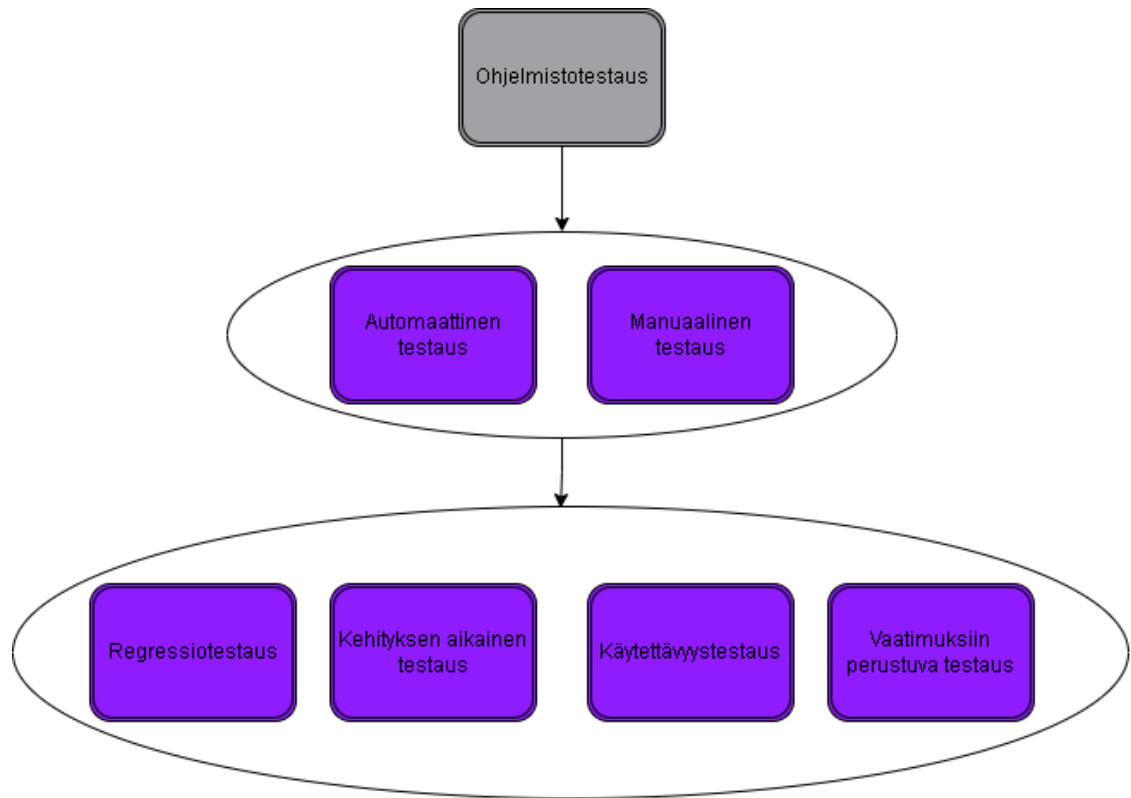
Kuva 2: Suurta kielimallia käyttävän generatiivisen tekoölyn kaaviokuva (Mohapatra et al., 2023) mukailten.

Ensimmäisenä kaaviokuvassa nähdään, kuinka suurta kielimallia esikoulutetaan käyttäen ohjaamatonta oppimista, suurta määrää tekstiä, miljardeja parametreja sekä eri algoritmeja. Esikoulutettu malli liitetään ohjelmoituun muuttajaan, josta syntyy generatiivinen esikoulutettu muuttaja. Generatiivista esikoulutettu muuttajaa käytetään itse generatiivisessa tekoälyohjelmassa, joka vastaanottaa erityyppisiä syötteitä ja tuottaa niiden perusteella tulosteita.

4. OHJELMISTOTESTAUS

Tässä luvussa käsitellään ohjelmistotestausta ja sen osa-alueita yleisellä tasolla ja perehdytään tarkemmin automaattiseen sekä manuaaliseen testaamiseen. Ensimmäisenä käsitellään ohjelmistotestausta yleisesti, jonka jälkeen käsitellään manuaalista testausprosessia. Viimeisenä käsitellään automaattista testausta ja mitä vaiheita testausprosessista voi automatisoida. Lisäksi esitellään luvussa 5 käytettävä testauksen prosessimalli.

Ohjelmistotestaus on prosessi, jossa asiakkaille pyritään toimittamaan virheetöntä ja luotettavaa ohjelmistoa (Khan, 2023). Ammann & Offutt (2016) mukaan ohjelmistotestauksen voi myös määritellä aktiviteetiksi, jossa varmistetaan ohjelman toimivan vaatimusten mukaisesti ajamalla ohjelma ja tutkimalla lopputulosta. Ohjelmistotestaus voidaan jakaa manuaaliseen testaamiseen ja automaattiseen testaamiseen (Kumar & Mishra, 2016). Manuaalisessa testaamisessa testitapaukset ajetaan käsin ja automaattisessa testaamisessa testitapaukset ajavat tietokoneet käyttäen ohjelmointikieliä apuna (Karhu et al., 2009). Sekä automaattisella testaamisella että manuaalisella testaamisella voidaan toteuttaa saman kategorian testejä, mutta automaattinen testaus on pääosin tehokkaampaa kuin manuaalinen testaus. Ohjelmistotestauksen voi jakaa kahteen eri osaan ja neljään kategoriaan, jotka ovat esitetty kuvassa 3. Tyypillisesti ohjelmistotestaamisen tavoitteena on löytää virheitä ja ongelmia ohjelmasta sekä varmistaa sidosryhmien vaatimusten täyttyminen (Olsen et al., 2018).



Kuva 3: Ohjelmistotestaamisen kaksi osaa ja neljä kategoriää (Job, 2021).

Tässä tutkimuksessa on tarkoituksena vain ymmärtää, mitä kuvassa 3 esitettyjen kategorioiden käsitteet tarkoittavat. Käsitteiden lyhyet määritelmät löytyvät taulukosta 2.

Taulukko 2: Testaamisen tyyppejä ja niiden määritelmiä

Vaatimukseen perustuva testaaminen

Tarkoituksena tarkistaa asiakkaan asettamien vaatimusten toteutuminen. Vaatimusten testaaminen tapahtuu hyväksymistestauksessa, joka on vaatimukseen perustuvan testaamisen viimeinen vaihe. Tämän jälkeen asiakas hyväksyy ohjelmiston varmistamalla, että hyväksymistestit oikein. (Job, 2021)

Käytettävyystestaus

Käyttöliittymä on oleellinen osa hyvää ohjelmistoa, sillä muutkin kuin kehittäjät käyttävät ohjelmistoa. Puutteellisen käyttöliittymätestaamisen seurauksena voi käyttöönoton jälkeen tapahtua kriittisiä virheitä. Käytettävyystestaus perustuu käyttöliittymän käyttämiseen, sille tarkoitetulla tavalla. Käytettävyystestausta voi tehdä joko kehittäjät, erilliset testaajat tai käyttäjäkokemusasiantuntija. (Job, 2021)

Kehityksen aikainen testaus

Testaamista toteutetaan kehittämisen aikana integraatio-, yksikkö- sekä järjestelmätesteinä. Prosessiin osallistuu koko kehitystiimi. (Job, 2021)

Regressiotestaus

Mitä tahansa testausta kehitys tai ylläpitovaiheessa, jossa varmistetaan, että uudet ominaisuudet tai korjaukset eivät riko aikaisemmin toimineita ominaisuuksia (Job, 2021).

4.1 Manuaalinen testaus

Manuaalitestauksessa ihminen ajaa ohjelmaa suorittaen eri testitapauksia. Testitapausten suorittamisen jälkeen tuloksia verrataan haluttuihin tuloksiin, jonka jälkeen testaajaa tekee tuloksista ohjelman laatua arvioivan raportin. (Kumar & Mishra, 2016) Manuaalisen testaamisen vaiheet riippuvat paljon henkilöstä, joka testaamista toteuttaa. Moiseev (2017) mukaan testaamisen vaiheet ovat seuraavat:

1. Testisuunnitelman tekeminen
2. Testitapausten suunnitteleminen
3. Savutestaus (engl. *smoke testing*)
4. Vikojen tarkastaminen
5. Regressiotestaus
6. Kuntotestaus (engl. *sanity testing*)
7. Testiraportin kirjoittaminen
8. Savutestit tuotannossa olevalla julkaisulla. (Moiseev, 2017)

Ensimmäinen testaamisen vaihe aloitetaan jo siinä vaiheessa, kun maininta mahdollisesta ohjelmistotuotteesta on tehty. Testaussuunnitelma sisältää testaajien määritelmän siitä mitä ohjelmaa testataan, mitä ohjelman toimintoja testataan, minkä tyyppisellä testaamisella ohjelmaa testataan, testauksen ehdot ja testauksen aloitus- ja lopetuskriteerit. (Moiseev, 2017)

Toinen vaihe alkaa testaajan saadessa järjestelmän määrittelyn ja dokumentaation. Testitapausten tulisi sisältää tapauksen vaiheet mahdollisimman tarkasti, odotettu lopputulos ja niiden tulisi perustua järjestelmän dokumentaatioon. (Moiseev, 2017) Testitapausten suunnittelu on kriittinen osa testaamista, sillä niiden avulla etsitään järjestelmästä vikoja. Huono testitapausten suunnittelu johtaa virheelliseen testaamiseen, jolloin oikeita virheitä ei välttämättä löydy.

Kolmas vaihe sisältää ensimmäiset testit, joita uudelle järjestelmän versiolle tehdään. Savutesteissä testataan järjestelmän perustoimintoja. Mikäli savutestien aikana löydetään ongelmia, kyseinen järjestelmän koontiversio todetaan epäonnistuneeksi ja ongelmia aletaan tutkimaan. (Moiseev, 2017)

Neljännessä vaiheessa tarkistetaan edellisissä versioissa ilmenneet viat, jotka ovat testaamista vaille valmiit. Nämä viat tulisi tarkistaa ensin, sillä jotkin korjaukset ovat voineet vaikuttaa muihin ominaisuuksiin. Mikäli testien aikana ilmenee uusia vikoja, merkataan ne ylös. (Moiseev, 2017)

Viides vaihe eli regressiotestaus aloitetaan savutestien ja vikojen tarkistamisen jälkeen. Regressiotesteissä käytetään toisessa vaiheessa tehtyjä testitapauksia. Regressiotestien tarkoituksena on tarkistaa koko järjestelmä vikojen varalta. Näiden testien aikana löytyneet viat merkataan ylös ja mikäli vika on kriittinen, tulee testaaminen aloittaa alusta eli vaiheesta kolme. (Moiseev, 2017)

Kuudes vaihe eli kuntotestaus toteutetaan tilanteissa, joissa julkaisua estävä tekijä huomataan liian myöhään eikä ole aikaa tehdä täyttä regressiotestausta. Kuntotestauksessa testataan vain korjattuun ominaisuuteen liittyvät asiat. Vaiheiden kuusi ja viisi jälkeen kirjoitetaan testausraportti, jossa todetaan, onko versio julkaisukelpoinen. (Moiseev, 2017)

Viimeisenä vaiheena toteutetaan savutestit uudestaan tuotannossa olevalla tuotteella. Tämän vaiheen tarkoituksena on testata ohjelmaa oikeilla käyttäjillä ja datalla. (Moiseev, 2017)

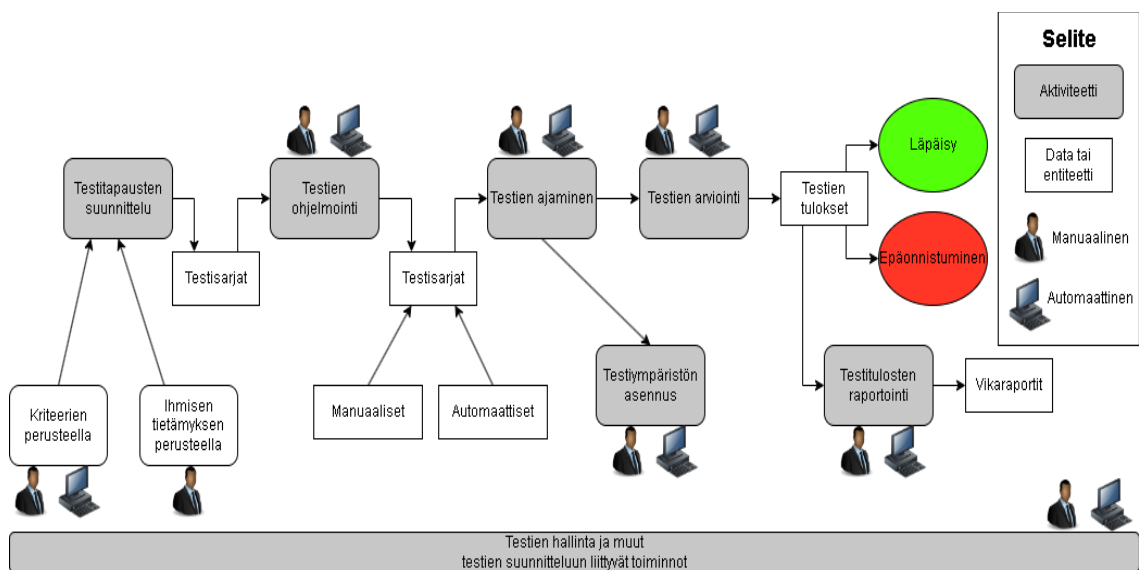
Yleisesti nämä vaiheet vaihtelevat paljon riippuen eri tekijöistä kuten ajasta ja projektista. Moiseev (2017) mukaan tämä malli on soveliaain ohjelman laadun mittaamiseen. Testaamisen onnistumisen takaamiseksi on testaajien ja koko tiimin noudatettava muutamia testaamiseen liittyviä sääntöjä. Ohjelmaan ei tulisi tehdä muutoksia testaamisen aikana, sillä pienikin muutos voi aiheuttaa ongelmia jo testatuissa ominaisuuksissa. Koko tiimillä tulisi olla täysi ymmärrys järjestelmästä, jotta testaaja ei ole aina sama henkilö. Samojen testien jatkuva ajaminen vaikeuttaa virheiden löytämistä, jonka takia testaajien tehtävien tulisi muuttua jatkuvasti. Viimeisimpinä sääntöinä ovat kaikille sopiva vikojen elinkaari sekä yhteinen pohja virheiden ja ongelmien hallintaan. (Moiseev, 2017)

4.2 Automaattinen testaus

Kuten luvun 4 alussa kerrottiin, automaattinen testaus on testitapausten ajamista tietokoneiden ja ohjelmointikielien avulla. Karhu et al. (2009) kertoo myös automaattisen testaamisen eli testiautomaation hyvistä ja huonoista puolista. Suurin hyöty automaattisesta testaamisesta on testaamisen nopeutuminen tietokoneiden tehokkaan suorituskyvyn ansiosta. Toisaalta automaattisen testaamisen merkittävänä huonona puolena on kasvavat kustannukset, sillä automaattinen testaaminen vaatii ylläpitoa kehittämistä sekä kouluttamista. (Karhu et al., 2009)

Testausprosessin vaiheista seuraavat ovat mahdollista automatisoida:

- Testitapausten suunnittelu
- Testien ohjelmointi
- Testien ajaminen
- Testien arviointi
- Testitulosten raportointi
- Testien hallinta ja muut testien suunnitteluun liittyvät toiminnot. (Garousi & Elberzhager, 2017)



Kuva 4: Testausprosessin automatisointimahdollisuudet (Garousi & Elberzhager, 2017) mukailten.

Testitapausten suunnittelun avulla saadaan luotua testisarjojen testitapaukset. Testitapausten suunnittelu voi perustua ihmisen tietämykseen tai valmiisiin kriteereihin. Täysin automatisoidun kriteeripohjaisen suunnittelu vaatii hakupohjaista testidatan luomista, jossa on mukana syötteet ja odotetut tulokset. (Garousi & Elberzhager, 2017)

Testien ohjelmointia on manuaalisten testien kirjoittamista tai automatisoitujen testisarjojen tekemistä testien ajoa varten. Testien ohjelmointia pystyy automatisoimaan osittain tai kokonaan. Osittainen automatisointi on mahdollista toteuttaa tallentamalla manuaalisen testaajan toteuttamia vaiheita testeistä, jonka jälkeen tietokone pystyy toistamaan ne. Täysi automaatio taas on mahdollista käyttämällä yhdistelmää luonnollisen kielen prosessoinnista, askelten takaperin ajosta, ajon aikainen tulkinnasta sekä oppimisesta. (Garousi & Elberzhager, 2017)

Testien ajaminen tarkoittaa testien suorittamista valitulla järjestelmällä. Testien ajon aikaiset tulokset tallennetaan tai järjestelmän toimintaa tarkastellaan ajon aikana. Testien ajaminen voi tapahtua täysin manuaalisesti, täysin automaattisesti tai osittain manuaalisesti ja automaattisesti. (Garousi & Elberzhager, 2017) Testien ajon aikaisen tarkastelun aikana tai testien ajon jälkeen arvioidaan testien tuloksia eli todetaan, onnistuivatko vai epäonnistuivatko testit. Testejä voi arvioida usealla eri tavalla riippuen, siitä ovatko testit automaattisia vai manuaalisia. Arviointi voi tapahtua ihmistestaajan omana arviona, automatisoidun ohjelman koodiin voidaan lisätä verifikaatiopisteitä, jotka varmistavat ulostulon oikeuden tai kehittäjät voivat rakentaa älykkäitä koneita, jotka tulkitsevat testejä, tekoälyn ja koneoppimisen avulla. (Garousi & Elberzhager, 2017)

Testitulosten raportointi on testaamisen viimeinen vaihe ja se sisältää testien lopputuloksen sekä testaamisen aikana löytyneet viat. Tämä vaihe on mahdollista toteuttaa automaattisesti, mutta sitä tehdään enemmän manuaalisesti. (Garousi & Elberzhager, 2017)

5. GENERATIIVINEN TEKOÄLY JA OHJELMISTO-TESTAUKSEN VAIHEET

Tässä luvussa käsitellään tutkimuksen alatutkimuskysymyksiä, jotka löytyvät luvusta 1.2. Aihetta käsitellään kuvan 4 mukaisen testausprosessin vaiheiden järjestyksessä. Tässä tutkimuksessa testaus luokitellaan manuaaliseksi, jos se sisältää mitä tahansa ihmisen tekemää työtä. Esimerkiksi pelkästään kehotteen eli syötteen antaminen tekoälyohjelmalle luokitellaan manuaaliseksi työksi.

5.1 Uusien testitapausten luominen

Aikaa vievä manuaalinen testitapausten kirjoittaminen on mahdollista korvata testitapausten automaattisella generoinnilla käyttäen eri menetelmiä kuten fuzz-testaus, ominaisuuksiin perustuva testaus (engl. *property-based testing*), hakupohjainen testaus (engl. *search-based testing*) ja kombinatorinen testaus. (Nie et al., 2023) Fuzz-testaus on virheiden etsimistä virheellisten syötteiden avulla (DeMott et al. 2018). Ominaisuuksiin perustuvassa testauksessa mallin määrittelyt ja invariantit ilmaistaan suoraan ohjelmakoodissa ja ne testataan automatisoidun ja satunnaistetun testidatan luomisen avulla (Thaler & Siebers 2020). Hakupohjainen testaus perustuu testidatan luomiseen kompleksisten algoritmien avulla (Klück et al. 2023). Kombinatorisessa testauksessa testitapauksia luodaan valitsemalla arvoja sisääntuloon ja yhdistämällä näitä arvoja (Lei et al. 2008). Nämä menetelmät ovat tehokkaita löytämään vikoja ohjelmasta, mutta sisältävät usein paljon tyylivirheitä, jonka takia testejä käytetään vain apuna manuaalisten testitapausten kirjoittamisessa. Testitapauksia voi myös generoida automaattisesti koneoppimisen avulla eli kouluttamalla koneoppimismallia olemassa olevilla testitapauksilla ja käyttämällä mallia uusien testien luomisessa. (Nie et al., 2023) Fan et al. (2023) mukaan käyttämällä hybridejä lähestymistapoja, jossa yhdistetään testien tuottamis- ja arviointitekniikoita sekä suuria kielimalleja, ovat tuottaneet lupaavia tuloksia. Nie et al. (2023) mukaan nykyaikaiset mallit pystyvät luomaan ihmisille ymmärrettävää koodia, mutta eivät pysty luomaan suurta määrää merkityksellistä koodia tai ymmärtämään laajempaa projektikontekstia. Testitapauksia on mahdollista luoda myös luonnollisella kielellä kirjoitettujen vikaraporttien perusteella, joka on juuri suurelle kielimallille sopiva tehtävä (Fan et al., 2023).

5.2 Testien ajaminen

Testien ajamiseen ei sellaisenaan voi implementoida generatiivista tekoälyä, sillä siihen kuuluu vain ympäristön pystyttäminen sekä skriptin ajaminen. Ajaminen tapahtuu pääosin automaattisesti, joko käyttäjän omassa tai jonkin pilvipalvelun ympäristössä. On kuitenkin mahdollista ohjelmoida ajettava testiskripti generatiivisen tekoälyn avulla hyödyntämällä ohjelmointikielen generointia (Wang et al., 2023). Schäfer et al. (2023) mukaan testien ajon aikana on mahdollista hyödyntää generatiivista tekoälyä epäonnistuvien testien kohdalla. Epäonnistuneet testit luovat kehotteen, joka sisältää epäonnistuneen testin sekä sen luoman virheviestin. Kehotteelle annetun tiedon avulla malli pystyy korjaamaan testin ajon aikana, jolloin testi taas onnistuu. (Schäfer et al., 2023) Tällä hetkellä Schäfer et al. (2023) ehdottamaa ideaa saisi parhaiten hyödynnettyä manuaalisen testien ajamisen aikana, sillä automaattinen ohjelmakoodin generointi sisältää vielä ongelmia kuten luvussa 5.1 on mainittu.

5.3 Testien sisään- ja ulostulo sekä niiden arviointi

Testioraakkeli on tiedon lähde sille, onko järjestelmän ulostulo oikea vai ei. Testioraakkeli voi olla esimerkiksi ohjelman toimintaa kuvaava käyttötapaus tai vaatimusmäärittelyssä ilmenevä laskentakaava. Näitä testioraakkeleita on myös mahdollista automatisoida generatiivisen tekoälyn avulla. Testien tehtävänä on varmistaa, että järjestelmän ulostulo on oikea. (Wang et al., 2023) Watson et al. (2020) mukaan monet nykyiset järjestelmät loivat merkityksettömiä assertioita eli väittämiä, jonka takia selkeäksi tavoitteeksi tuleville järjestelmille on luoda merkityksellisiä väittämiä.

Wang et al. (2023) mukaan suuria kielimalleja alettiin käyttämään väittämien luonnissa suorituskyvyn parantamiseksi, sillä takaisinkytketyillä neuroverkoilla vain 17 prosenttia generoiduista väittämistä vastasi totuutta. Suurien kielimallien käyttäminen selkeästi nostaa generoinnin suorituskkyä ja luonnollisella ja ohjelmointikielellä koulutettu malli nosti suorituskvyn 57 prosenttiin (Wang et al., 2023). Nashid et al. (2023) saavutti hyödyntämällä minimaalista datamäärää eli harvaa oppimista noin 76 prosentin tarkkuuden väittämiin. Generoidut testit usein sisältävät turhia assertointeja, jonka takia generoinnin jälkeen tulisi minimoida assertointien määrää käyttäen olemassa olevia työkaluja (Dakhel et al., 2023).

Testien sisääntulo on myös mahdollista automatisoida generatiivisen tekoälyn avulla. Automaattisesti generoidut sisääntulot ovat hyvä yhdistää testioraakkeliin kanssa, jotta testien onnistumisen arviointi olisi mahdollista (Wang et al., 2023). Testien sisääntulo

vaihtelee ohjelmien välillä ja koostuu erilaisista tekstisyötteistä sekä toimintojen yhdistelmistä (Wang et al., 2023). Schäfer et al. (2023) mukaan generatiivinen tekoäly ja suuret kielimallit soveltuvat paremmin regressiotestien luomiseen kuin vikojen löytämiseen.

Tässä tutkimuksessa käsitellään vain tekstimuotoisten syötteiden generointia. Tekstimuotoisten syötteiden luominen on hankalaa myös ihmiselle, koska tekstisyötteet voivat olla useita eri muodoissa ilmaistavia asioita (Liu et al., 2023). Liu et al. (2023) mukaan kehitteillä olevat suuret kielimallit, jotka ovat koulutettu erittäin laajamittaisilla korpuksilla, pystyvät ymmärtämään hankaliakin syötteitä ja tuottamaan oikeanlaisen syötteen mallin ulostulona.

5.4 Testitulosten raportointi ja vianmääritys

Testien aikana löytyneet viat raportoidaan testien ajon päätyttyä. Vikoja pystytään raportoimaan automaattisesti luonnollisella kielellä generatiivisen tekoälyn avulla, jossa suuri kielimalli on koulutettu käyttäen aikaisempien vikojen korjausten kuvauksia (Mahbub et al., 2023). Vikojen otsikointia voi myös automatisoida vikojen kuvauksen perusteella käyttämällä generatiivista tekoälyä (Zhang et al., 2022).

Vianmäärityksessä etsitään virheen lähde sekä korjataan löytyneet viat. Vianmääritystä voi tehdä käyttäen suuria kielimalleja. Bui et al. (2022) esittelee tutkimuksessaan viitekehystä, jossa suurella kielimallilla tunnustaa viallisen koodin osan ja virheelliset rivit, jonka jälkeen muuttaa virheelliset rivit toimiviksi. Kang et al. (2023) taas esittelee tutkimuksessaan viitekehystä, jossa suurta kielimallia käytetään vikojen uudelleenluomiseen ja viikoja toistavien testitapausten ehdottamiseen kehittäjälle vianmäärityksen helpottamiseksi.

Vikojä voidaan myös korjata käyttäen suuria kielimalleja ja generatiivista tekoälyä. Lajkó et al. (2022) tutkimuksessa käsitellään yksinkertaisinta tapausta, jossa kielimallilla pyritään korjaamaan yhden koodirivin vikoja. Oikeissa tilanteissa ilmenevät bugit ovat kuitenkin yleensä kompleksisempia ja sisältävät useampia rivejä koodia, jonka takia uudemmat tutkimukset ovat keskittyneet kompleksisempiin tilanteisiin (Wang et al., 2023). Wang et al. (2023) mukaan vikoja voidaan korjata käyttäen generatiivista tekoälyä myös luonnollisella kielellä kirjoitetun kuvauksen sekä staattisen analysointin avulla. Näitä ei kuitenkaan tässä tutkimuksessa käydä tarkemmin läpi, koska ne eivät kuulu tutkimuksen rajausalueeseen.

6. YHTEENVETO

Tämän luvun tarkoituksena on tiivistää tutkimuksessa tehdyt havainnot, esitellä niistä tehdyt johtopäätökset sekä jatkotutkimusehdotuksia. Luku sisältää myös itserefleksointia tutkimuksen toteutuksesta ja sen onnistumisesta.

6.1 Tutkimuksen tulokset

Tässä tutkimuksessa perehdyttiin tekoölyyn, suuriin kielimalleihin, ohjelmistotestaukseen sekä generatiivisen tekoölyn hyödyntämiseen ohjelmistotestausprosessissa. Tutkimuksen tarkoituksena oli vastata luvussa 1.2 asetettuun tutkimuskysymykseen:

- Miten generatiivista tekoölyä hyödynnetään ohjelmistotestauksessa?

Sekä alatutkimuskysymyksiin:

- Miten generatiivista tekoölyä hyödynnetään ohjelmistotestauksen manuaalisessa testaamisessa?
- Miten generatiivista tekoölyä hyödynnetään ohjelmistotestauksen automaattisessa testaamisessa?

Tutkimukseen löytyi erittäin rajallinen määrä lähteitä ja monet lähteet ovat julkaistu tämän vuoden aikana eikä niitä ole välttämättä vertaisarvioitu. Yksittäin generatiivisesta tekoölystä ja ohjelmistotestauksesta kuitenkin löytyi hyvin vertaisarvioituja lähteitä.

Tutkimuksen tulos oli kirjallisuuden ajankohtaisuuden ansiosta selkeä. Ensimmäiseen alatutkimuskysymykseen ei kuitenkaan saatu selkeää vastausta, sillä kaikki lähdeaineisto käsitteli pääosin automatisointia. Tutkimuksen perusteella generatiivista tekoölyä voi hyödyntää manuaalisessa testaamisessa automatisoimalla testausprosessia osittain ja käyttämällä generatiivista tekoölyä keskustelukumppanina. Automaattisen testaamisen kannalta useat testausprosessin osat olivat mahdollista automatisoida. Testitapauksia on jossain määrin mahdollista luoda ja suunnitella, mutta kielimallit eivät vielä pysty ymmärtämään suuria projektikonaisuuksia eli tämä prosessi vaatii ihmisen valvomista. Testien ajamiseen liittyen ei löytynyt mitään merkityksellistä käyttöä generatiiviselle tekoölylle. Testien sisään- ja ulostuloon sekä niiden arviointiin löytyi useita keinoja hyödyntää generatiivista tekoölyä, jotka olivat testioraakkelieneroointi, tekoölyllä luotujen testien turhien assertointien minimointi ja tekstimuotoisten syötteiden generointi. Tutkimuksessa todettiin myös, että generatiivinen tekoöly sekä suuret kielimallit soveltuvat pa-

remmin regressiotestien luomiseen kuin vikojen löytämiseen. Testien raportointiin liittyen löytyi myös useita generatiiviseen tekoälyyn liittyviä mahdollisuuksia kuten vikojen automaattinen raportointi aikaisempia vikaraportteja hyödyntävä kielimallin avulla sekä vikojen otsikointi hyödyntämällä vian kuvausta. Testien vianmääritykseen löytyi myös useita generatiivista tekoälyä hyödyntävää keinoa kuten viallisen koodin osan löytäminen sekä sen korvaaminen toimivalla koodilla ja vikojen uudelleenluomista generatiivisella tekoälyllä, jonka jälkeen kehittäjälle suositellaan vikoja toistavia testitapauksia. Vikojen korjaamista ei laajemmin tutkittu, sillä se ei kuulunut tutkimuksen rajausalueeseen. Kaikki lähdeaineistossa esitellyt menetelmät vaikuttivat kuitenkin olevan vielä kehitysvaiheessa, joten ohjelmistotestauksen täyttä automaatiota ei ole vielä mahdollista saavuttaa. Taulukosta 3 löytyy generatiivisen tekoälyn käyttöön liittyviä hyötyjä ja ongelmia ohjelmistotestauksessa.

Taulukko 3: Generatiivisen tekoälyn hyödyt ja ongelmat ohjelmistotestauksessa.

Testausvaihe	Hyödyt	Ongelmat
Uusien testitapausten luominen	Vähentää merkittävästi automaattiseen testaukseen kohdistuvaa manuaalista työpanosta. On mahdollista luoda testitapauksia ohjelmakoodina tai luonnollisena kielenä.	Monet testitapaukset voivat olla tarpeettomia, sillä kielimallit eivät ole riittävän kehittyneitä. Kielimallit eivät ymmärrä laajaa projektikontekstia eivätkä pysty luomaan suurta määrää merkityksellistä ohjelmakoodia.
Testien ajaminen	Testiskriptien automaattinen luonti nopeuttaa valmiiden testien siirtämistä automaattiseen ajoon. Testejä on mahdollista korjata ajon aikana nopeuttaen testausprosessia.	Testien ajon aikainen ongelmien korjaaminen mahdollista vain, mikäli testien ajo tapahtuu manuaalisesti.
Testien sisään- ja ulostulo ja niiden arviointi	Ulostulon assertointien ja sisään tulon automaattinen luominen nopeuttaa automaattista testausprosessia	Nykyiset järjestelmät luovat merkityksettömiä assertioita. Assertointien

	poistamalla manuaalisia vaiheita.	luominen ei auta manuaalisessa testausprosessissa. Syötteitä luovat kielimallit ovat vasta kehitteillä.
Testitulosten raportointi ja vianmääritys	Raportointi nopeutuu ja löydettyjen vikojen selkeytyy, jolloin vikojen jatkotoimenpiteet nopeutuvat. Ohjelmakoodissa olevia vikoja on mahdollista tunnistaa ja korjata.	Raportit ja otsikot voivat olla virheellisiä, jolloin jatkotoimenpiteisiin kuluu merkittävästi enemmän aikaa. Kompleksisempia tilanteita ei ole vielä mahdollista korjata.

6.2 Jatkotutkimusehdotukset

Generatiivisen tekoälyn tutkimus on erittäin tuoretta ja monet siihen liittyvät mahdollisesti tämän tutkimuksen kannalta hyödylliset tutkimukset ovat kesken. Tässä tutkimuksessa teoriaa käsiteltiin pintapuolisesti, jonka takia esimerkiksi tekoälymallien matemaattinen teoria jäi käsittelemättä. Jatkotutkimuksessa teoriaa esiteltäisiin vielä kattavammin ja erilaisia ohjelmistotestaukseen ja generatiiviseen tekoälyyn liittyviä viitekehyksiä sekä menetelmiä voitaisiin vertailla niiden toiminnan ja taloudellisen kannattavuuden kannalta. Paul (2023) mukaan generatiivinen tekoäly on vielä kokeellisessa vaiheessa ja vertaa tätä itseohjautuviin ajoneuvoihin, jotka toimivat jossain määrin ja eivät vielä ole riittävän luotettavia ihmisen korvaamiseen. Sauer (2023) uutisartikkelista nähdään, että muutama maailman johtavista teknologiayrityksistä ovat vasta tämän vuoden aikana julkaisseet ensimmäisiä tekoälytyökaluja. Jatkotutkimuksen kannalta olisi siis aiheellista odottaa muutama vuosi, jotta laadukasta lähdeaineistoa löytyy riittävästi. Vikojen korjaamiseen voisi myös perehtyä tarkemmin jatkotutkimuksessa.

LÄHTEET

- Ammann, P., Offutt, J., 2016. Introduction to software testing.
- Brandoni, N., 2023. The Evolution of AI: Exploring Its Impac and Potential.
- Bui, N.D.Q., Wang, Y., Hoi, S., 2022. Detect-Localize-Repair: A Unified Framework for Learning to Debug with CodeT5. <https://doi.org/10.48550/ARXIV.2211.14875>
- Dakhel, A.M., Nikanjam, A., Majdinasab, F., Desmarais, M.C., 2023. Effective Test Generation Using Pre-trained Large Language Models and Mutation Testing. <https://doi.org/10.48550/ARXIV.2308.16557>
- DeMott, J. et al. (2018) Fuzzing for software security testing and quality assurance. Second edition. Boston, Massachusetts: Artech House.
- Dobrev, D., 2012. A Definition of Artificial Intelligence. <https://doi.org/10.48550/ARXIV.1210.1568>
- Euchner, J., 2023. Generative AI. Res.-Technol. Manag. 66, 71–74. <https://doi.org/10.1080/08956308.2023.2188861>
- Fan, A., Gokkaya, beliz, Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., Zhang, J.M., 2023. Large Language Models for Software Engineering: Survey and Open Problems. <https://doi.org/10.48550/ARXIV.2310.03533>
- Fink, A., 2014. Conducting Research Literature Reviews, 4th ed, From the Internet to Paper. Thousand Oaks (Calif.): Sage, Los Angeles.
- Garousi, V., Elberzhager, F., 2017. Test Automation: Not Just for Test Execution. IEEE Softw. 34, 90–96. <https://doi.org/10.1109/MS.2017.34>
- Goericke, S., 2020. The future of Software Quality Assurance, 1st ed. Cham: Springer Nature.
- Hooda, S., 2023. Agile software development: trends, challenges and applications. Hoboken, New Jersey: John Wiley & Sons.
- Hu, Z., Xing, E.P., 2022. Toward a “Standard Model” of Machine Learning. Harv. Data Sci. Rev. <https://doi.org/doi:10.1162/99608f92.1d34757b>
- IEEE, 1990. IEEE. <https://doi.org/10.1109/IEEESTD.1990.101064>
- IEEE 730, 2014. IEEE Standard for Software Quality Assurance Processes.
- ISO/IEC, 1995. Information technology — Vocabulary — Part 28: Artificial intelligence — Basic concepts and expert systems.
- Job, M.A., 2021. Automating and Optimizing Software Testing using Artificial Intelligence Techniques. Int. J. Adv. Comput. Sci. Appl. 12. <https://doi.org/10.14569/IJACSA.2021.0120571>

Kang, S., Yoon, J., Yoo, S., 2023. Large Language Models are Few-shot Testers: Exploring LLM-based General Bug Reproduction, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Presented at the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, Melbourne, Australia, pp. 2312–2323. <https://doi.org/10.1109/ICSE48619.2023.00194>

Kang, Y., Cai, Z., Tan, C., Huang, Q., Liu, H., 2020. Natural language processing (NLP) in management research: A literature review. *J. Manag. Anal.* 7, 139–172. <https://doi.org/10.1080/23270012.2020.1756939>

Karhu, K., Repo, T., Taipale, O., Smolander, K., 2009. Empirical Observations on Software Testing Automation. Presented at the International Conference on Software Testing Verification and Validation, Denver, CO, USA, 201–209. <https://doi.org/10.1109/ICST.2009.16>

Khan, R., 2023. Deep Learning System and It's Automatic Testing: An Approach. *Ann. Data Sci.* 10, 1019–1033. <https://doi.org/10.1007/s40745-021-00361-w>

Khanagar, S.B., Al-ehaideb, A., Maganur, P.C., Vishwanathaiah, S., Patil, S., Baeshen, H.A., Sarode, S.C., Bhandi, S., 2021. Developments, application, and performance of artificial intelligence in dentistry - A systematic review. *J. Dent. Sci.* 16, 508–522.

Klück, F. et al. (2023) An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems. *Information and software technology*. [Online] 160107225-.

Kumar, D., Mishra, K.K., 2016. The Impacts of Test Automation on Software's Cost, Quality and Time to Market. *Procedia Comput. Sci.* 79, 8–15. <https://doi.org/10.1016/j.procs.2016.03.003>

Lajkó, M., Csuvik, V., Vidács, L., 2022. Towards JavaScript program repair with generative pre-trained transformer (GPT-2), in: Proceedings of the Third International Workshop on Automated Program Repair. Presented at the ICSE '22: 44th International Conference on Software Engineering, ACM, Pittsburgh Pennsylvania, 61–68. <https://doi.org/10.1145/3524459.3527350>

Lee, J., Suh, T., Roy, D., Baucus, M., 2019. Emerging Technology and Business Model Innovation: The Case of Artificial Intelligence 5, 44. <https://doi.org/10.3390/joitmc5030044>

Lei, Y. et al. (2008) IPOG/IPOG-D: efficient test generation for multi-way combinatorial testing. *Software testing, verification & reliability*. [Online] 18 (3), 125–148.

Liu, Z., Chen, C., Wang, J., Che, X., Huang, Y., Hu, J., Wang, Q., 2023. Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Presented at the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, Melbourne, Australia, 1355–1367. <https://doi.org/10.1109/ICSE48619.2023.00119>

Mahbub, P., Shuvo, O., Rahman, M.M., 2023. Explaining Software Bugs Leveraging Code Structures in Neural Machine Translation, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Presented at the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, Melbourne, Australia, pp. 640–652. <https://doi.org/10.1109/ICSE48619.2023.00063>

Mohapatra, D.P., Thiruvoth, F.M., Tripathy, S., Rajan, S., Vathulya, M., Lakshmi, P., Singh, V.K., Haq, A.U., 2023. Leveraging Large Language Models (LLM) for the Plastic Surgery Resident Training: Do They Have a Role? *Indian J. Plast. Surg.* s-0043-1772704. <https://doi.org/10.1055/s-0043-1772704>

Moiseev, D.A., 2017. METHODOLOGY AND PROCESS OF MANUAL TESTING. *Reliab. Qual. Complex Syst.* <https://doi.org/10.21685/2307-4205-2017-3-16>

Nadkarni, P.M., Ohno-Machado, L., Chapman, W.W., 2011. Natural language processing: an introduction. *J. Am. Med. Inform. Assoc.* 18, 544–551. <https://doi.org/doi.org/10.1136/amiajnl-2011-000464>

Nashid, N., Sintaha, M., Mesbah, A., 2023. Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Presented at the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, Melbourne, Australia, 2450–2462. <https://doi.org/10.1109/ICSE48619.2023.00205>

Nie, P., Banerjee, R., Li, J.J., Mooney, R.J., Gligoric, M., 2023. Learning Deep Semantics for Test Completion, in: 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE). Presented at the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), IEEE, Melbourne, Australia, pp. 2111–2123. <https://doi.org/10.1109/ICSE48619.2023.00178>

Niittyviita, A., 2023. Nopea ohjelmistotestaus.

Olsen, K., Parveen, T., Black, R., Friedenber, D., Hamburg, M., McKay, J., Posthuma, M., Schaefer, H., Smilgin, R., Smith, M., Toms, S., Ulrich, S., Walsh, M., Zakaria, E., 2018. Foundation Level Syllabus. *Int. Softw. Test. Qualif. Board.*

Osterweil, L.J., 2008. What is software? *Autom. Softw. Eng.* 15, 261–273. <https://doi.org/10.1007/s10515-008-0031-y>

Paul, K., 2023. Generative AI still mostly experimental, say executives. *Reutersolse*

Samoili, S., Lopez Cobo, M., De Prato, G., Martinez-Plumed, F., Delipetrev, B., 2020. AI Watch Defining Artificial Intelligence. Towards an operational definition and taxonomy of artificial intelligence (No. EUR 30117 EN). Publications Office of the European Union.

Sauer, C., n.d. AWS, Google Cloud, Microsoft Azure: Who's leading the game in Generative AI? *Medium.*

Schäfer, M., Nadi, S., Eghbali, A., Tip, F., 2023. An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. <https://doi.org/10.48550/ARXIV.2302.06527>

Schulmeyer, G.G., 2007. *Handbook of software quality assurance*, 4th ed. Boston Artech House.

Thaler, J. & Siebers, P.-O. (2020) Specification testing of agent-based simulation using property-based testing. *Autonomous agents and multi-agent systems.* [Online] 34 (2), .

Wang, J., Huang, Y., Chen, C., Liu, Z., Wang, S., Wang, Q., 2023. Software Testing with Large Language Model: Survey, Landscape, and Vision. <https://doi.org/10.48550/ARXIV.2307.07221>

Watson, C., Tufano, M., Moran, K., Bavota, G., Poshyvanyk, D., 2020. On learning meaningful assert statements for unit test cases, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering. Presented at the ICSE '20: 42nd International Conference on Software Engineering, ACM, Seoul South Korea, 1398–1409. <https://doi.org/10.1145/3377811.3380429>

Zannier, C., Erdogmus, H., Lindstrom, L., 2004. Extreme Programming and Agile Methods - XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods, Lecture Notes in Computer Science. Calgary.

Zhang, T., Irsan, I.C., Thung, F., Han, D., Lo, D., Jiang, L., 2022. iTiger: an automatic issue title generation tool, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Presented at the ESEC/FSE '22: 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM, Singapore Singapore, 1637–1641. <https://doi.org/10.1145/3540250.3558934>