Ilona Juvonen

Ohjelmistotestaus ja tekoälyjärjestelmien testauksen erityispiirteet

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

Tiivistelmä

Tekijä:	Ilona Juvonen
Otsikko:	Ohjelmistotestaus ja tekoälyjärjestelmien testauksen erityispiirteet
Sivumäärä:	35 sivua
Aika:	30.4.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Jorma Räty

Tässä insinöörityössä tarkasteltiin ohjelmistotestauksen perusperiaatteita ja haasteita erityisesti tekoälyjärjestelmien näkökulmasta. Keskeisiksi haasteiksi ja perinteisten- sekä tekoälyjärjestelmien testauksen eroavaisuuksiksi havaittiin mallien epädeterministinen käyttäytyminen, datan laatuun ja edustavuuteen liittyvät ongelmat sekä päätöksenteossa esiintyvät vinoumat, jotka vaikuttavat myös järjestelmän eettisyyteen ja turvallisuuteen.

Ratkaisuina näihin ongelmiin esiteltiin datakeskeinen testaus, toleranssipohjaiset arviointikriteerit, metamorfinen testaus sekä mallin yleistettävyyttä arvioiva ristiinvalidointi. Lisäksi vinoumien tunnistamiseksi ja eettisyyden arvioimiseksi esiteltiin tilastollisia analyysejä sekä selittäviä algoritmeja. Työssä korostettiin, että tekoälyjärjestelmien testaus ei pääty järjestelmän käyttöönottoon, vaan se vaatii jatkuvaa ylläpitoa ja seurantaa esimerkiksi mallin ajautumisen vuoksi.

Alalta puuttuu vielä yhtenäinen testausmenetelmien standardointi, mikä vaikeuttaa laadun vertailua ja arviointia. EU:n tekoälyasetus (AI Act) luo sääntelykehikon tekoälyn käytölle erityisesti korkean riskin sovelluksissa ja vahvistaa tarvetta osoittaa tekoälyjärjestelmien turvallisuus, läpinäkyvyys ja oikeudenmukaisuus myös testauksen avulla. Tulevaisuudessa tekoälyn testauskäytäntöjen yhtenäistämiseen tarvitaan lisää tutkimusta sekä alan yhteistyötä.

Avainsanat: Ohjelmistotestaus, tekoälyjärjestelmien testaus, testidatan laatu, vinoumat, testioraakkeli

---

Abstract

Author:	Ilona Juvonen
Title:	Software Testing and the Specific Characteristics of AI System Testing
Number of Pages:	35 pages
Date:	30 April 2025
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Development
Supervisor:	Jorma Räty, Senior Lecturer

This thesis examines the fundamental principles of software testing spesically from the perspective of artificial intelligence (AI) systems. The key challenges identified include managing the non-deterministic nature of AI models, ensuring the quality and representativeness of test data, and addressing biases in decision-making, which also impact the ethical and safety aspects of AI systems.

Proposed solutions include data-centric testing, tolerance-based acceptance criteria, metamorphic testing, and cross-validation to evaluate model generalization. Statistical analysis and explainability techniques are presented to identify biases and assess ethical considerations. Additionally, it was emphasized that AI system testing does not conclude upon deployment but requires ongoing monitoring and maintenance due to phenomena such as concept drift.

However, a significant issue within the field remains the lack of standardized testing methodologies, complicating consistent quality assessment across systems. The EU's AI Act provides a regulatory framework, especially for high-risk AI applications, reinforcing the need for transparency, safety, and fairness, aspects that must be verifiably demonstrated as well through testing. Future research and collaborative industry efforts are necessary to develop standardized and effective testing practices for AI systems.

Keywords: Software testing, testing of AI systems, quality of test data, biases, test oracle

Sisällys

Lyhenteet

SDLC: Software Development Life Cycle on ohjelmistokehityksen elinkaari, joka kuvaa ohjelmistokehityksen prosessit vaihe vaiheelta.

STLC: Software Testing Life Cycle on ohjelmistotestauksen elinkaari. Mukailee ohjelmistokehityksen elinkaaren vaiheita testauksen näkökulmasta.

F1-arvo: F1-arvo on koneoppimismallin mittari, joka yhdistää tarkkuuden ja herkkyyden yhteen arvoon. Se kuvaa mallin kokonaissuorituskykyä erityisesti silloin, kun luokkien jakauma on epätasainen. F1-arvo on korkea vain silloin, kun malli suoriutuu hyvin sekä tarkkuudessa että herkkyydessä.

Herkkyys: Herkkyys mittaa, kuinka monta todellisista positiivisista tapauksista malli osasi tunnistaa oikein. Herkkyys lasketaan jakamalla oikein tunnistetut positiiviset oikein tunnistettujen positiivisten ja väärien negatiivisten summalla.

Tarkkuus: Tarkkuus mittaa, kuinka moni mallin positiiviseksi luokittelema tapaus oli oikeasti positiivinen. Se lasketaan jakamalla oikein tunnistetut positiiviset oikein tunnistettujen positiivisten ja väärien positiivisten summalla.

1 Johdanto

Ohjelmistotestaus on olennainen osa ohjelmistokehityksen elinkaarta. Testauksen avulla pyritään varmistaa ohjelmiston toimivuus, laatu ja luotettavuus ennen tuotantoon viemistä. Perinteisten ohjelmistojen testausprosessit ovat vakiintuneet ja hyvin dokumentoituja, mutta teknologian kehittyessä ja tekoälyn yleistyessä ohjelmistoissa myös testaamisen vaatimukset ja lähestymistavat ovat muuttumassa.

Tekoälypohjaiset sovellukset asettavat testaukselle uusia vaatimuksia. Näissä järjestelmissä testauksen tavoitteena ei ole ainoastaan tekninen virheettömyys, vaan myös mallin käyttäytymisen ennakoitavuus, oikeudenmukaisuus, eettisyys ja turvallisuus. Perinteisistä ohjelmistoista poiketen tekoälyjärjestelmät perustuvat itsenäisesti toimiviin malleihin ja suurten datamäärien käsittelyyn, mikä tekee niiden toiminnasta osittain ennakoimatonta ja testauksen tulkinnasta haastavampaa. Tämä kehitys vaatii ohjelmistotestauksen ammattilaisilta uudenlaista ajattelua sekä laajempaa ymmärrystä datan laadusta, mallien epädeterministisyydestä ja algoritmien mahdollisista vinoumista.

Opinnäytetyö toteutetaan yhteistyössä AliceAI Learning Oy:n kanssa, joka kehittää tekoälyä hyödyntävää kielenoppimissovellusta eri ammattiryhmien tarpeisiin, minkä vuoksi on tarpeellista tarkastella, miten tekoälypohjaisten järjestelmien testaus eroaa perinteisestä ohjelmistotestauksesta ja mitä erityispiirteitä se tuo mukanaan.

Tämän insinöörityön tavoitteena on tarkastella sekä perinteistä ohjelmistotestausta että tekoälyä hyödyntävien järjestelmien testausta teoreettisella tasolla ja tuoda esille niiden keskeiset erot ja erityispiirteet. Työssä hyödynnetään kirjallisuuskatsausta ja standardeja lähteinä. Työssä tutkitaan kysymyksiä: Mitkä ovat tekoälyä hyödyntävien ohjelmistojen testaamisen merkittävimmät haasteet? Miten perinteisten ohjelmistojen ja tekoälyjärjestelmien testaus eroavat toisistaan?

2 Yleistä ohjelmistotestauksesta

Ohjelmistotestaus on keskeinen kiinteä osa ohjelmistokehitystä, jonka tarkoituksena on varmistaa, että ohjelmisto toimii suunnitellusti ja täyttää sille asetetut vaatimukset. Ilman huolellista testausta ohjelmistot voivat sisältää virheitä, jotka heikentävät käyttäjäkokemusta, aiheuttavat tietoturvariskejä tai voivat johtaa merkittäviin taloudellisiin menetyksiin. Ohjelmistotestauksella ei kuitenkaan voida todistaa ohjelmiston täydellistä virheettömyyttä, vaan sillä voidaan todistaa ainoastaan, että ohjelmistossa on löydettyjä tai piilossa olevia virheitä [1]. Tässä luvussa käsitellään ohjelmistotestauksen merkitystä ja sen perusperiaatteita.

2.1 Testaamisen merkitys

Ohjelmistotestauksen ajatellaan joskus olevan ylimääräinen investointi, jonka merkitystä moni ei ymmärrä. Tämä voi johtua siitä, että useinkaan ei tiedetä, mitä kaikkea ohjelmistotestaus pitää sisällään, ja sen ajatellaan tuovan lisätyötä, joka ei tuota mitään. On kuitenkin tutkittu, että testaamaton tuote voi nostaa kehityskustannuksia huomattavasti. Esimerkiksi Yhdysvalloissa arvioidaan vuonna 2022 huonon ohjelmistolaadun kustannusten olevan noin 2,41 biljoonaa dollaria. Kyseiset kustannukset johtuvat esimerkiksi teknisestä velasta, kyberrikollisuudesta ja ohjelmistojen toimitusketjun ongelmista. [2.] Lisäkustannuksia tuo myös jälkikäteen tehtävä virheiden korjaus, joka olisi ollut halvempaa tehdä huolellisen testauksen myötä aikaisessa vaiheessa. Tästä syystä ohjelmistotestaus on ylimääräisen investoinnin sijaan *oleellinen* investointi, joka voi vähentää kehityskustannuksia merkittävästi etenkin pitkällä aikavälillä.

Testaaminen on merkittävää myös käyttäjätyytyväisyyden ja -turvallisuuden vuoksi. Testauksella voidaan varmistaa, että tuote toimii toivotulla tavalla ja asiakas osaa käyttää ohjelmistoa intuitiivisesti. Hyvin testattu sovellus ja käyttöliittymä lisää asiakastyytyväisyyttä ja luo luottoa yritystä kohtaan. Käyttäjäkokemuksen lisäksi käyttäjän tulee pystyä luottamaan siihen, että hänen henkilötietonsa eivät voi levitä sovelluksen ulkopuolelle. Huolellisella testauksella voidaan

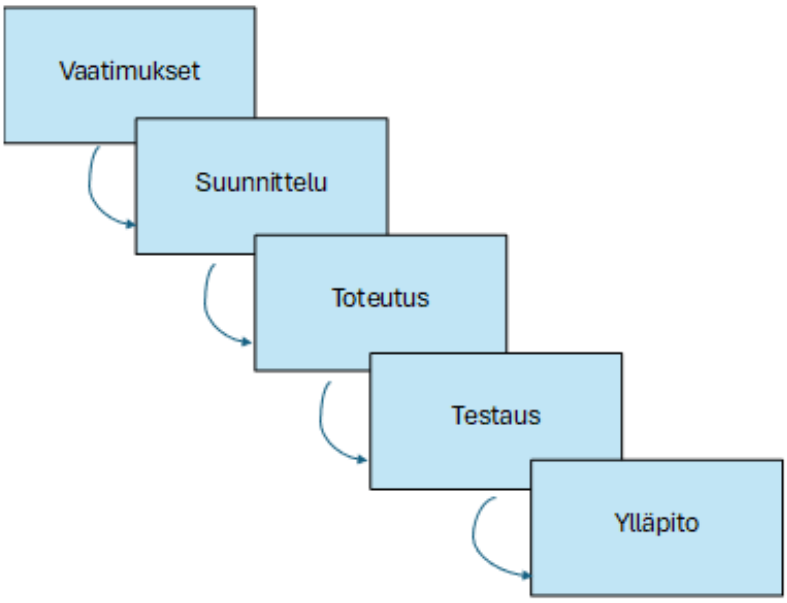
suojata myös käyttäjien fyysinen terveys. Varoittavia esimerkkejä on useita, mutta yksi merkittävä huolellisella testauksella estettävissä oleva ohjelmistovirhe tapahtui vuosina 1985–1987 käytetyssä Therac25-tietokoneohjatussa sädehoitolaitteessa, joka annosteli ohjelmavirheen vuoksi osalle potilaille satakertaisia annosmääriä sädehoitoa aiheuttaen näin vakavan loukkaantumisen tai kuoleman [3].

Syyt ohjelmistotestauksen hyödyistä kumuloituvat lopulta yhteen käsitteeseen *laatu*. Mikään tuote ei menesty markkinoilla pitkäkestoisesti ilman huolellista laadunvarmistusta. Laadukas tuote on laadukas niin käyttäjän kuin kehittäjän näkökulmasta. Käyttäjien oletuksena on saada laadukas tuote, jota on helppo käyttää ja joka toimii odotetulla tavalla. Mikäli käyttäjällä herää epäluottamusta tuotetta kohtaan, tai hän huomaa tuotteen käytössä puutteita, hän voi siirtyä käyttämään toista vastaavaa tuotetta, jolloin käyttäjiltä tuleva rahavirta katkeaa. Kehittäjien näkökulmasta laadukas ohjelmisto takaa hyvän maineen ja markkinoilla menestymisen lisäksi myös sen, että yllättäviä kustannuksia jälkikäteen tehtävistä virheenkorjauksista ei tule.

2.2 Testaus ohjelmistokehityksessä

Ohjelmistotestaus on kehittynyt merkittävästi vuosikymmenten saatossa, ja sen rooli ohjelmistokehityksessä on muuttunut kehityksen loppuvaiheessa tehtävästä virheiden etsimisestä osaksi kokonaisvaltaista laadunvarmistusta. Ohjelmistokehityksessä käytetään molempia testausta ja virheenkorjausta laadunvarmistuksessa. Isoimmat erot näiden kahden välillä ovat niiden prosessit - testauksessa tunnistetaan ja etsitään ohjelmistovirheitä ja virheenkorjauksessa korjataan jo tunnistettuja virheitä [4].

Perinteinen ohjelmistokehitys tapahtui ennen pääosin Kuva 1 näkyvän vesiputousmallin mukaisesti, jossa kehitys etenee lineaarisesti vaiheesta toiseen.



Kuva 1 Ohjelmistoprojektin vesiputousmalli

Vesiputousmallissa testaus tapahtuu vasta kehityksen loppuvaiheessa, mikä saattaa johtaa suuriin muutostarpeisiin, jos virheitä löydetään myöhään. Mallin ongelmana on joustamattomuus, sillä muutosten tekeminen on kallista ja aikaa vievää, ja kehityksessä taaksepäin meno on vaikeaa. Testauksen näkökulmasta tämä tarkoittaa sitä, että virheiden korjaaminen on hidasta, eikä jatkuvaa laadunvarmistusta voida toteuttaa tehokkaasti. [5.] Tämä johti testauksen siirtymiseen kehityksen alkuvaiheisiin, missä testauksesta tuli olennainen osa ohjelmiston elinkaarta. Nykyisin testaus on ennakoivaa ja integroitua, ja se nähdään laadunvarmistuksen välineenä, joka tukee ohjelmistokehityksen kaikkia vaiheita

Agile-mallissa ohjelmistokehitys tapahtuu iteratiivisesti ja yhteistyöhön perustuen. Agile-metodit, kuten Scrum ja Kanban, painottavat jatkuvaa parantamista ja tiivistä yhteistyötä kehitystiimien ja sidosryhmien välillä. Agile-mallissa testaus on jatkuvaa, ja se tapahtuu rinnakkain kehityksen kanssa. Tämä mahdollistaa nopean palautteen ja virheiden varhaisen havaitsemisen, mikä vähentää kehityskustannuksia ja parantaa lopputuotteen laatua. [5.]

Testivetoisessa kehityksessä (TDD) ohjelmakoodia kirjoitetaan testien pohjalta. Ensin kehittäjä laatii testitapauksen, joka määrittelee, miten ohjelman osan tulisi toimia. Vasta tämän jälkeen varsinaista ohjelmakoodia kirjoitetaan niin, että testit menevät läpi. Tämä iteratiivinen prosessi auttaa varmistamaan ohjelmiston laadun ja vähentää virheiden määrää jo kehitysvaiheessa.

Nykyisin ohjelmistotestauksessa korostuvat ennaltaehkäisevät menetelmät, kuten testivetoisen kehityksen ja käyttäjäkeskeisen testauksen merkitys. Laadunvarmistus ei ole enää vain ohjelmistokehityksen loppuvaiheen tehtävä, vaan se on integroitu osaksi koko kehitysprosessia, mikä takaa paremman ja kestävämmän ohjelmistokehityksen [6].

2.3 Ohjelmistotestauksen haasteet

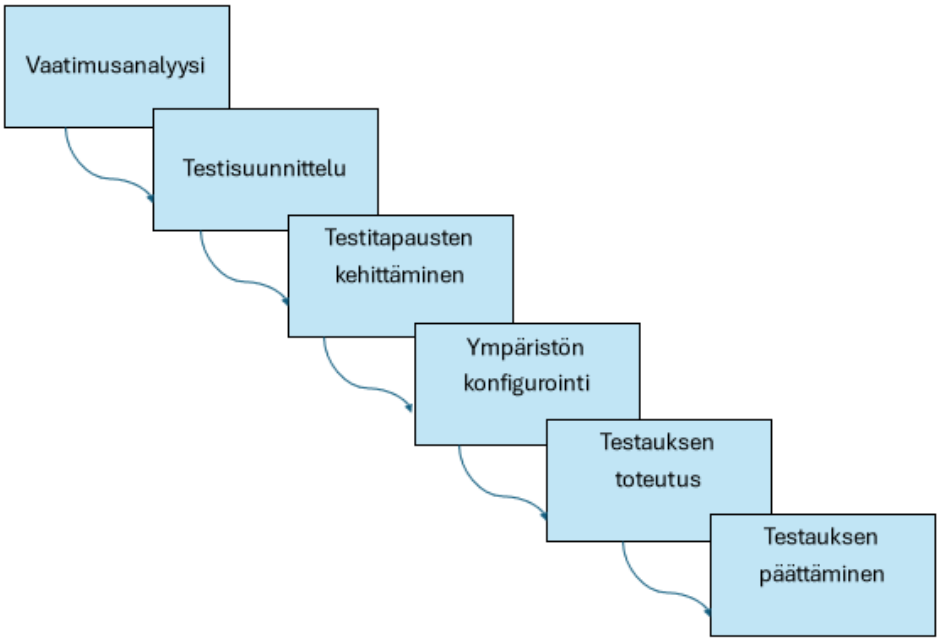
Ohjelmistotestauksessa on monia haasteita, jotka yleensä painottuvat erityisesti ei-teknisiin tekijöihin, kuten tiimien väliseen yhteistyöhön ja prosessien hallintaan. Yksi suurimmista ongelmista on kommunikaation puute tiimien välillä, mikä voi aiheuttaa viivästyksiä ja heikentää ohjelmistojen laatua.

Laajan ohjelmiston testauksessa kaikkien mahdollisten testitapausten suorittaminen voi olla mahdotonta aikataulu- ja resurssirajoitteiden vuoksi ja ohjelmistoja julkaistaan jatkuvasti monien virheiden kanssa. On tärkeää priorisoida huolellisesti testattavat alueet riskien ja liiketoiminnan tarpeiden perusteella, sillä virheellinen priorisointi voi johtaa kriittisten virheiden huomaamatta jäämiseen, mikä heikentää ohjelmiston laatua.

Testausympäristöt voivat myös olla epävakaita, mikä hankaloittaa testaamista. Kehittäjät tekevät muutoksia testausympäristöihin esimerkiksi korjatakseen virheitä tai lisätäkseen ominaisuuksia. Muutokset, joita kehittäjät tekevät ympäristöissä, eivät aina heti saavuta QA-tiimiä, mikä voi johtaa epäselvyyksiin, kuten vanhan version testaamiseen ja jo korjattujen virheiden virheilmoitusten toistumiseen testauksessa [7].

3 Ohjelmistotestauksen elinkaari

Ohjelmistotestauksen elinkaari STLC (Software Testing Life Cycle) on ohjelmistotestauksen prosessi, joka liittyy vahvasti myös ohjelmistokehityksen elinkaareen (SDLC), mikä kuitenkin pitää sisällään vain testauksen vaiheet. Se on Kuva 2 esitetty systemaattinen prosessi, johon kuuluu kuusi eri vaihetta, joista kaikilla on oma tarkoitus ja päämäärä sekä määritellyt sisääntulo- ja poistumiskriteerit, toiminnot ja toimitukset. STLC alkaa heti, kun ohjelmistovaatimukset on määritelty. Tekoälypohjaisten ohjelmistotestauksen elinkaari noudattaa samaa kaavaa ja kriteereitä kuin perinteinen ohjelmistotestaus, mutta jokaisessa vaiheessa painotetaan tekoälyn erityispiirteitä ja sen vaikutusta testausprosessiin.



Kuva 2 Ohjelmistotestauksen elinkaari

Vaatimusanalyysi on STLC:n ensimmäinen vaihe, jossa testitiimi vaiheen nimen mukaisesti analysoi ohjelmistovaatimukset ja pyrkii ymmärtämään, mitä ja miten näitä vaatimuksia voidaan testata [1]. Tässä vaiheessa on tärkeä olla selkeä

kommunikaatio testitiimin, kehittäjien sekä sidosryhmien välillä, jotta lähtökohta on kaikille sama. Tämä vaihe on keskeinen, sillä se määrittelee testauksen suuntaviivat ja varmistaa, että testaus kattaa kaikki olennaiset liiketoiminta- ja tekniset vaatimukset**.** Vaatimusanalyysivaiheessa, jossa perinteiset sovellukset keskittyvät toiminnallisiin ja ei-toiminnallisiin vaatimuksiin, tekoälypohjaisissa sovelluksissa painotetaan myös datan laatuvaatimuksia ja mallin suorituskyvyn mittareita, kuten tarkkuus, muistaminen ja selitettävyys. [8.]

STLC:n toisessa vaiheessa testisuunnittelussa laaditaan testausstrategia, valitaan testausmenetelmät ja määritellään testaukseen tarvittavat resurssit. Suunnitteluvaiheessa päätetään myös testauksen kattavuudesta, testauksen lopetuskriteereistä sekä mahdollisista testauksen automatisointityökaluista. [1.] Tekoälypohjaisissa sovelluksissa suoritetaan samat vaiheet kuin perinteisessä ohjelmistotestauksessa, mutta lisänä testaajat tunnistavat myös keskeisiä suorituskykyindikaattoreita (KPI) arvioidakseen tekoälymallin tarkkuutta, nopeutta ja tehokkuutta. Sisääntulokriteereinä ovat hyväksytyt vaatimusmäärittelyt ja aiemmin tunnistetut riskit, ja poistumiskriteereinä ovat valmis testisuunnitelma ja hyväksytyt testausstrategiat. [9.]

Kun suunnitelma on valmis, testitapausten kehittämisvaiheessa laaditaan yksityiskohtaiset testitapaukset, jotka kattavat erilaiset skenaariot ohjelmiston koodatusta toiminnallisuudesta. Tässä vaiheessa määritellään myös testidata, jota käytetään testien suorittamiseen. [9.] Tekoälyn testaukseen liittyy ainutlaatuisia haasteita perinteiseen ohjelmistotestaukseen verrattuna, joten tämän vaiheen tavoitteena on varmistaa, että kaikki mahdolliset skenaariot, mukaan lukien reunatapaukset, on katettu, sekä toteuttaa mallin ja datan testaus [8]. Tämä vaihe on tärkeä, sillä se määrittelee koko käytännön testausprosessin alun. Sisääntulokriteereinä on hyväksytty testisuunnitelma ja määritellyt vaatimukset, ja poistumiskriteerinä ovat valmiit testitapaukset ja testidatasetit.

Testauksen onnistuminen on riippuvainen myös siitä, että testiympäristö on oikein konfiguroitu. Testiympäristön asennuksessa varmistetaan, että ohjelmiston testaamiseen käytettävät laitteistot, ohjelmistot ja työkalut on asennettu oikein

[10]. Tekoälyjärjestelmiä testattaessa on hyvä ottaa huomioon laitteistojen suorituskyky, sillä tekoälyjärjestelmät vaativat usein tehokkaan laitteiston laskelmien suorittamiseen. Testausympäristössä tulee olla tarvittavat resurssit sekä oikeat tietojoukot (datasets) tekoälymallien koulutusta, testausta ja validointia varten. Tämä vaihe hyvin suoritettuna mahdollistaa luotettavan ja toistettavan testauksen. Sisääntulokriteereinä ovat testattavan ohjelmiston valmis versio ja ympäristön määritykset. Poistumiskriteerinä on onnistunut testiympäristön käyttöönotto.

Kun testiympäristö on valmis, testit voidaan suorittaa. Testauksen toteutusvaiheessa kehitetyt testitapaukset ajetaan, tulokset analysoidaan ja mahdolliset virheet raportoidaan kehitystiimille. Tarvittaessa virheiden korjaamisen jälkeen suoritetaan regressiotestauksia, jotta voidaan varmistaa, että korjaukset eivät ole aiheuttaneet ohjelmistoon uusia ongelmia. Tekoälyjärjestelmän testauksen toteutus sisältää ohjelmiston toiminnallisuuden testauksen lisäksi myös mallin arvioinnin ja ennustusten validoinnin. Tekoälyjärjestelmissä tulosten analysointi ja regressiotestaus on monimutkaisempaa ja voi kestää pidempään kuin perinteisissä sovelluksissa. Sisääntulokriteereinä ovat valmiit testitapaukset, testiympäristö ja testidatasetit, ja poistumiskriteereinä ovat suoritetut testit ja raportoitu testitulokset. [1.]

Viimeisessä vaiheessa testausprosessi päätetään ja laaditaan testausraportti, jossa arvioidaan testauksen onnistuminen ja tunnistetut haasteet. Kaikki tunnistetut viat tai ongelmat pitää olla ratkaistu tässä vaiheessa. Raportti sisältää yhteenvedon löydetyistä virheistä, testauksen tehokkuudesta ja mahdollisista suosituksista tuleviin testausprosesseihin. Tekoälyjärjestelmien testauksen päätösraporttiin kirjataan erityisesti mallin suorituskyvyn jatkuva seuranta, kuten mallin heikkeneminen (drift) ja uudelleenkoulutustarpeet. Lisäksi raporttiin voidaan merkitä suosituksia mallin ja datan edustavuuden parantamiseksi. Sisääntulokriteereinä ovat kaikki aiemmat testausvaiheet ja analysoidut tulokset, ja poistumiskriteerinä on hyväksytty testiraportti ja loppuarviointi testauksen kattavuudesta. [9.]

STLC-prosessi tapahtuu toistuvasti ohjelmistokehityksen aikana, ei vain kerran. Tämä johtuu siitä, että ohjelmistokehitys on iteratiivinen prosessi, erityisesti Agile-menetelmien mukaisesti, joissa uusia ominaisuuksia lisätään jatkuvasti. Jokaisessa kehityssyklissä, kuten Sprintissä tai julkaisussa, STLC käydään läpi uudelleen varmistaen, että uudet muutokset eivät riko aiempia toiminnallisuuksia ja että ohjelmisto toimii odotetusti. Lisäksi regressiontestaus ja jatkuva laadunvarmistus edellyttävät STLC:n vaiheiden toistamista aina, kun ohjelmistoon tehdään muutoksia.

4 Ohjelmistotestauksen tasot

Ohjelmistotestaus voidaan pääosin jakaa neljään progressiiviseen tasoon, jotka ovat yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. Nämä neljä tasoa kuvaavat ohjelmistokehityksen vaiheita testauksen aikana. Kuva 3 esitetään testauksen tasot ja niitä vastaavat ohjelmistokehityksen vaiheet.

Kuva 3 Ohjelmistotestauksen tasot. Kuvassa mukailtu nettisivun Geeksfor-Geeks kuvaa testaustasoista.

Kaikilla näillä tasoilla on omat tehtävänsä ja merkityksensä, ja ne tulee suorittaa huolellisesti ja järjestyksessä. [11.]

4.1 Yksikkötestaus

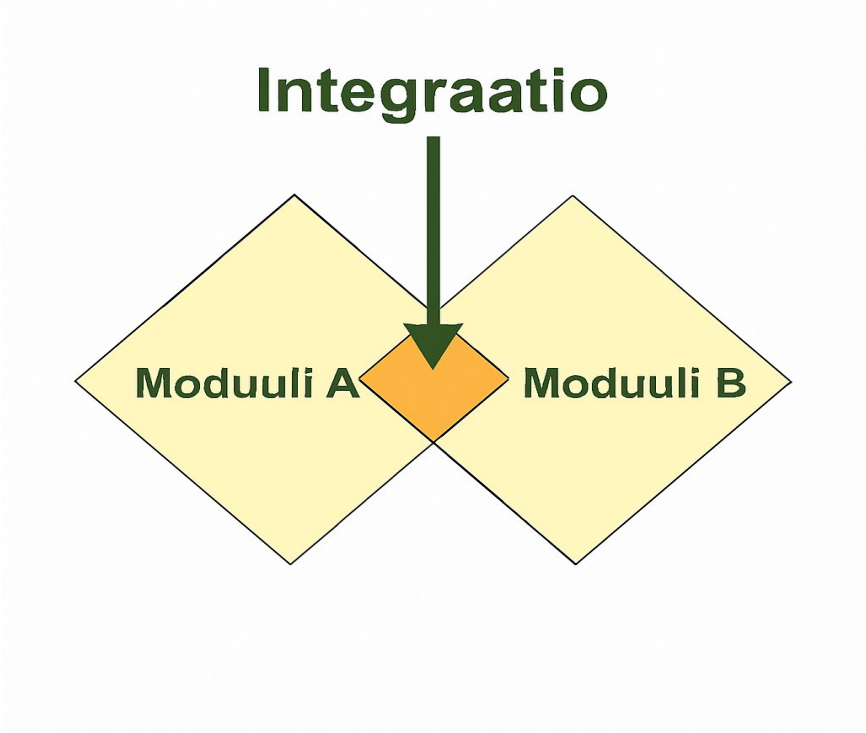
Yksikkötestaus on ohjelmistotestauksen perusta, jonka testauskohteena on yhden yksittäisen funktion, luokan tai moduulin toiminta. Yleensä toiminnallisuuden kehittäjä tekee testauksen, koska yksikkötestauksessa varmistetaan uuden toiminnon toimivuus ennen kuin se integroidaan osaksi muita järjestelmän osia. Tyypillisesti tässä testausvaiheessa virheitä tai läpi menemättömiä testejä ei raportoida, sillä mahdollisen virheen löytyessä kehittäjä voi korjata virheen saman tien. Näin uusien toiminnallisuuksien kehitys ja testaus kulkevat näppärästi käsi kädessä, mikä pienentää samalla tulevaisuuden virheilmoitusten riskiä. [1]

Yksikkötestauksessa on kuitenkin haasteita, kuten se, että useat komponentit toimivat vuorovaikutuksessa keskenään eivätkä itsenäisesti, jolloin niiden toiminnallisuutta voi olla haasteellista testata kokonaisuudessaan. Sen vuoksi testauksen avuksi toteutetaan usein testikomponentteja, joiden avulla simuloidaan komponenttien vuorovaikutusta [12]. Muut haasteet liittyvät resurssien, ajan ja osaamisen puutteeseen, sillä jokainen ohjelmiston komponentti täytyy testata erikseen ja se vie aikaa.

Yksikkötestauksen hyöty on se, että koodissa olevat virheet on helppo jäljittää, ne löytyvät ja niiden korjaus tapahtuu varhaisessa vaiheessa, ennen kuin ne pääsevät vaikuttamaan muihin komponentteihin ja ohjelmiston toimintaan. Yksikkötestit auttavat koodin ylläpidossa ja varmistavat yksittäisten komponenttien tai toiminnallisuuksien laadun. Yksikkötestaus myös auttaa virheenkorjauksessa ja paikantamisessa. Virheiden korjaamisen kustannukset ovat alhaiset, koska virheet löytyvät ajoissa - jos yksikkötesti epäonnistuu, vain viimeisimpiä muutoksia tarvitsee tarkastella. Vaikka yksikkötestauksen rooli laadunvarmistuksessa on korvaamaton, se ei yksinään voi varmistaa ohjelmiston laadukkuutta, vaan tarvitsee kaverikseen muita testausmenetelmiä.

4.2 Integraatiotestaus

Yksikkötestauksen jälkeen erilliset ohjelmistokomponentit yhdistetään ja niiden yhteistoiminta varmistetaan. Kuva 4 havainnollistetaan ohjelmistotestauksen toista tasoa integraatiotestausta, jossa testataan kahden, tai useamman, jo yksikkötestatun ohjelmistokomponentin tai -moduulin kanssakäymistä. Tämän tason tavoitteena on löytää mahdolliset rajapintaongelmat, väärin toteutetut integraatiot ja tietovirheiden käsittelyyn liittyvät puutteet, sekä pienentää riskejä. [1]



Kuva 4 Havainnollistava kuva integraatiotestauksesta

Yksi suurimmista haasteista integraatiotestauksessa on testausympäristön luominen. Testausta varten tarvitsee luoda ympäristö, joka simuloi todellista tuotantoa mahdollisimman tarkasti, jotta voidaan testata useiden komponenttien välistä vuorovaikutusta mahdollisimman oikeanmukaisesti. Tämä voi olla aikaa vievää sekä monimutkaista, varsinkin, jos järjestelmässä on useita ulkoisia palveluja, joiden kanssa sovelluksen tulee toimia. Tämän lisäksi virheiden jäljittäminen on haasteellisempaa kuin yksikkötestauksessa, sillä virhe voi sijata missä tahansa järjestelmän osassa, ja se voi esiintyä vain tietyissä olosuhteissa tai

tiettyjen komponenttien yhdistelmillä. Tähän auttaa myös ensimmäisellä tasolla tehty huolellinen yksikkötestaus, jolla voidaan varmistaa, että yksittäiset komponentit toimivat odotetusti.

Integraatiotestaus on välttämätöntä ja tuo parempaa luotettavuutta ohjelmistoon. Integraatiotestit varmistavat, että eri komponentit toimivat luotettavasti yhdessä ja minimoi riskiä mahdollisten ongelmakohtien siirtymisestä seuraavalle tasolle. Integraatiotestaus on ohjelmistotestauksen toiseksi tärkein vaihe yksikkötestauksen jälkeen, sillä se toimii ikään kuin siltana yksikkötestauksen ja laajemman kolmannen tason järjestelmätestauksen välissä.

4.3 Järjestelmätestaus

Järjestelmätestaus on ohjelmistokehityksen vaihe, jossa koko integroitu järjestelmä testataan. Sen tavoitteena on arvioida järjestelmän yhteensopivuutta ennalta määriteltyjen vaatimusten kanssa. Tämä testaus suoritetaan *black box* -menetelmällä, mikä tarkoittaa, että testaaja ei näe koodin sisäistä toteutusta tai logiikkaa. Laatuasiantuntijat luottavat spesifikaatioihin ja varmistavat, että järjestelmä toimii odotetulla tavalla. [1.]

Järjestelmätestaukseen liittyy monia samoja haasteita kuin yksikkö- ja integraatiotestaukseen, mutta suuremmassa mittakaavassa. Koska testattava järjestelmä on paljon laajempi, myös aika-, kustannus- ja kompleksisuushaasteet kasvavat. Erityisesti järjestelmäintegraatiotestauksessa testataan eri järjestelmien ja palveluiden välisiä rajapintoja, mikä tehdään yleensä järjestelmätestauksen yhteydessä tai sen jälkeen. Tähän voi sisältyä myös ulkopuolisten organisaatioiden, kuten verkkopalveluiden, liitännät, mikä lisää testauksen monimutkaisuutta. Kun testattava palvelu ei ole täysin organisaation hallinnassa, voi virheiden korjaaminen ja testausympäristöjen hallinta muodostua haasteelliseksi. [1.]

Järjestelmätestauksen tavoitteena on varmistaa koko ohjelmiston toimivuus ennen sen siirtymistä seuraavaan testausvaiheeseen. Testausympäristön tulee

vastata mahdollisimman tarkasti tuotanto- tai liiketoimintaympäristöä, jotta löydetyt virheet heijastaisivat todellisia käyttötilanteita. Testit suoritetaan erilaisilla testiskripteillä, jotka kattavat sekä tekniset että liiketoiminnalliset vaatimukset. Järjestelmätestauksen jälkeen iso osa ohjelmiston bugeista on löydetty ja kehittäjät voivat jatkaa luottavaisesti hyväksymistestauksen pariin.

4.4 Hyväksymistestaus

Hyväksymistestaus on ohjelmistotestauksen neljäs taso, joka eroaa testausmenetelmältään aiemmista tasoista keskittyessään lopullisten asiakkaiden ja käyttäjien näkökulmaan ilman syventymistä ohjelmistokoodiin [13]. Tämän tason tavoitteena on varmistaa, että ohjelmisto täyttää vaaditut kriteerit ja selventää ratkaisua, onko ohjelmisto valmis julkaistavaksi.

Hyväksymistestaustyyppejä on useita erilaisia, joista jokaisella on omat tavoitteensa. Testauksella voidaan esimerkiksi varmistaa, että ohjelmisto vastaa tehtyä sopimusta ja lakisääteisiä vaatimuksia, tai arvioida järjestelmän operatiivista valmiutta, kuten varasuunnitelmia ja ylläpitoprosesseja.

Alpha- ja Beta-testaustyypit puolestaan keskittyvät erityisesti ohjelmiston käytettävyyteen ja viimeistelyyn ennen julkaisua. Alpha-testaus suoritetaan kehittäjän tai sisäisen testausryhmän toimesta kontrolloidussa ympäristössä, ja sen tavoitteena on havaita kriittiset virheet ennen ohjelmiston laajempaa jakelua. Beta-testaus puolestaan tapahtuu todellisilla loppukäyttäjillä heidän omissa käyttöympäristöissään, jolloin saadaan arvokasta palautetta ohjelmiston toiminnasta ja mahdollisista kehityskohdista ennen lopullista julkaisua. [14.]

Hyväksymistestauksen haasteet liittyvät edeltävien tasojen mukaisesti ajallisiin haasteisiin. Varsinaisilla käyttäjillä tehtävät testit edellyttävät, että käyttäjiä on tarpeeksi. Välillä testikäyttäjiä on vaikea saada ja sitouttaa, mikä vaikeuttaa ja pitkittää palautteen saantia. Lisäksi käyttäjäkokemukset ja mielipiteet voivat vaihdella suuresti, joten tarvitaan riittävä määrä palautetta kriittisimpien kehityskohteiden tunnistamiseksi. Silti hyväksymistestaus on keskeinen osa

ohjelmistokehitystä, sillä se varmistaa, että tuote vastaa käyttäjien odotuksia ja täyttää vaaditut standardit ennen julkaisua.

5 Tekoälypohjaiset järjestelmät ja testauksen haasteet

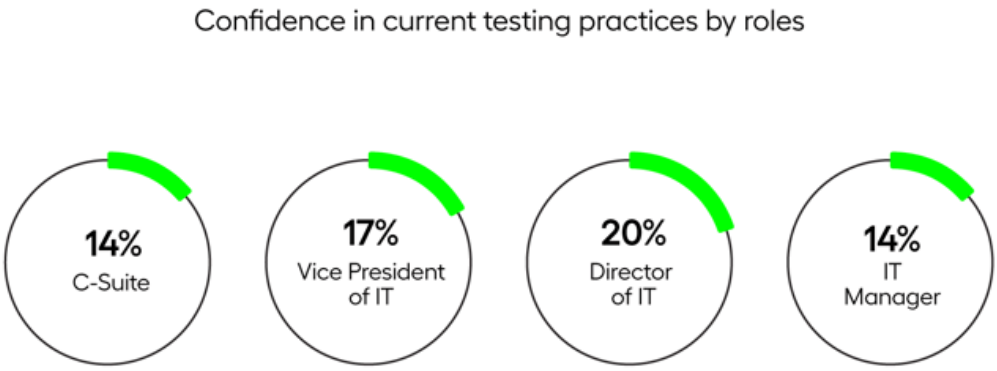
Tekoälypohjaiset ohjelmistoratkaisut ja -komponentit ovat yleistyneet huomattavasti viime vuosien aikana, ja niiden kehitys tulee jatkumaan nousujohteisesti. Tekoälyjärjestelmät käyttävät algoritmeja, jotka analysoivat dataa, tunnistavat kaavoja ja tekevät päätöksiä ilman, että niiden toimintaa on suoraan ohjelmoitu. Tässä työssä tekoälyjärjestelmällä tarkoitetaan yläkäsitettä, joka kattaa yleisesti erilaiset tekoälyjärjestelmät ja -teknologiat. Tyypillisimpiä tekoälyteknologioita ovat kuitenkin koneoppiminen (Machine Learning, ML), syväoppiminen (Deep Learning, DL), vahvistusoppiminen (Reinforcement Learning, RL) ja generatiiviset mallit, joista koneoppiminen ja generatiiviset mallit nousevat tässä työssä useimmin esille [15].

Koneoppiminen on tekoälyn keskeinen osa-alue, jossa kehitetyt algoritmit oppivat datasta ja tekevät ennusteita tai luokituksia ilman erillistä suuntaa antavaa ohjelmointia. Koneoppimisella on muutamia alahaaroja, kuten syväoppiminen ja vahvistusoppiminen. Syväoppiminen hyödyntää monikerroksisia neuroverkkoja ja soveltuu kuvantunnistukseen ja luonnollisen kielen käsittelyyn. Vahvistusoppimista hyödynnetään muun muassa robotiikassa ja pelitekoälyssä, ja se perustuu kokeilemalla oppimiseen. Generatiiviset tekoälymallit, kuten esimerkiksi ChatGPT, pystyvät tuottamaan uutta tekstiä ja luomaan kuvia ja muuta mediaa opitun datan pohjalta.

Tekoälyjärjestelmiin liittyy vahvasti datan käsittely, sillä niille on ominaista datavetoisuus. Käytännössä tämä tarkoittaa sitä, että tekoälyjärjestelmien suorituskykyyn, eettisyyteen ja laatuun vaikuttaa merkittävästi koulutusdatan laatu ja kattavuus. Tekoälyjärjestelmät mukautuvat ja kehittyvät syöttödatan perusteella toisin kuin perinteiset ohjelmistot, jotka perustuvat ennalta määriteltyihin sääntöihin. Tämä tekee tekoälypohjaisten järjestelmien toiminnasta ja sen testauksesta monimutkaista, minkä vuoksi tekoälypohjaisten järjestelmien testaaminen

on suosittu ja ajankohtainen tutkimuskohde. Tekoälyjärjestelmien testaamiseen ei ole vielä vakiintunut yleisiä käytäntöjä, mikä johtaa epäjohdonmukaisiin arviointi- ja validointiprosesseihin [16].

Raportissa “AI and Software Quality: Trends and Executive Insights” tulee ilmi, että vain 16 % tutkimukseen osallistuneista neljästäsadasta yrityksestä uskoo, että heidän nykyiset testausmenetelmänsä ovat tarpeeksi tehokkaita [17]. Tutkimuksessa eroteltiin erikseen myös teknologia-alan ammattilaisten luottosuhde nykyisiin tekoälyjärjestelmien testausmenetelmiin, joita Kuva 5 on esitelty. Tuloksista nähdään tyhjentävästi, että kenenkään ammattiryhmän luottoprosentti ei ole yli 20 %, joka viittaa selkeästi siihen, etteivät testausmenetelmät ole vielä riittävän kattavat tai luotettavalla tasolla.

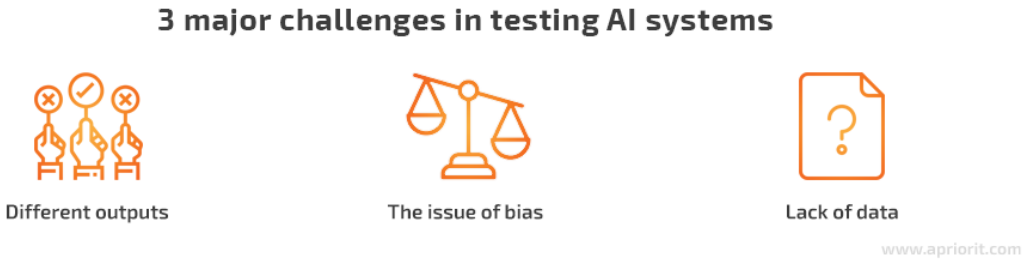


Kuva 5 Ammattiryhmien luottamusprosentit nykyisiin tekoälyn testausmenetelmiin [17]

5.1 Tekoälyjärjestelmien testauksen yleisimmät haasteet

Tekoälyjärjestelmiä on kehitetty moniin erilaisiin tarpeisiin, ja niiden toimintalogiikka ja arkkitehtuuri vaihtelevat riippuen kyseisen järjestelmän tarpeista. Vaikka haasteet ovat pääosin järjestelmäriippuvaisia ja testausmenetelmiä tulee

soveltaa testattavaan tekoälyjärjestelmään, on kuitenkin kolme yleistä päähaastetta, jotka toistuvat alalla kirjoitetuissa tutkimuksissa ja joita kehittäjät ja testaajat tulevat kohtaamaan riippumatta siitä, minkälaista tekoälyjärjestelmää he ovat kehittämässä. Yleisimmät haasteet on esitelty Kuva 6.



Kuva 6 Yleiset tekoälyjärjestelmien testaushaasteet [18]

5.1.1 Mallien epädeterministisyys

Yksi keskeisistä haasteista tekoälyjärjestelmien testauksessa liittyy tekoälymallien epädeterministiseen ja todennäköisyysperusteiseen luonteeseen. Tekoälymallien epädeterministinen ja todennäköisyysperustainen luonne aiheuttavat erityisiä haasteita testauksen suunnitteluun ja toteutukseen. Epädeterministisyydellä tarkoitetaan sitä, ettei tekoälymalli välttämättä tuota aina samaa lopputulosta samoilla syötteillä ja todennäköisyysperustaisella sitä, että mallin voidaan odottaa tuottavan hyväksytty vastaus tietyin todennäköisyysperustein.

Perinteisessä ohjelmistotestauksessa testit oletetaan toistettaviksi – samanlaisten testiajojen odotetaan tuottavan samanlaiset tulokset, mikä helpottaa odotetun ja toteutuneen lopputuloksen vertailua. Tekoälyjärjestelmissä mallien sisäinen satunnaisuus ja itsenäiseen oppimiseen liittyvät tekijät voivat aiheuttaa vaihtelua saaduissa tuloksissa ja niiden käyttäytymisessä. Kun mallit oppivat ajan saatossa itsenäisesti, on todennäköistä, että aiemmin läpi menneet testit eivät mene läpi enää mallin päivittyneemmässä versiossa. Testit, joita malli ei päässyt läpi, voi olla helppo jäljittää, mutta uusille toiminnallisuuksille testien generointi on haasteellista. Lisäksi mallia testatessa tulee huomioida riski, että malli oppii uudesta testauksesta epähaluttuja asioita. [19.]

Testioraakkelilla tarkoitetaan lähdettä, joka tietää tehdyn testitapauksen odotetun vastauksen. Testioraakkeli voi olla esimerkiksi vertailtava järjestelmä, käyttötapaus tai yksittäisen henkilön erityistieto. [20.] Perinteisessä ohjelmistossa testioraakkelin toteuttaminen on yleensä helppoa, sillä kuten edellä mainittiin, jokaiselle testitapaukselle voidaan määrittää odotettu lopputulos. Epädeterministisissä ja todennäköisyyksiin pohjautuvissa tekoälymalleissa tilanne on usein haastavampi, koska sama syöte voi tuottaa erilaisia, vaikkakin hyväksyttäviä tuloksia eri suorituskerroilla [21]. Tämä tekee yhden selkeän lopputuloksen määrittämisestä hankalaa, tai jopa mahdotonta, jonka vuoksi tekoälyjärjestelmien testauksessa testioraakkelien ja niiden hyväksymiskriteerien on oltava joustavampia. Tuloksille asetetaan usein yksittäisen hyväksytyn arvon sijaan hyväksymisalue tai toleranssiraja, joka huomioi mallin tuottamien tulosten luonnollisen vaihtelun. Tämä kuvastaa laajempaa oraakkeliongelmaa, jossa haasteena on määritellä luotettavasti ja yhdenmukaisesti mallin tuloksen hyväksymisrajat, eli milloin tulos on tarpeeksi lähellä odotettua tulosta ollakseen hyväksyttävä. [19.]

5.1.2 Vinoumat

Ihmisiä ja tekoälyä yhdistävät ajattelun vinoumat (biases). Vinoumia on monia erilaisia, ja kaikilla meistä on joitakin niistä. Yksi yleisistä vinoumista on ”minulla ei ole vinoumaa” -vinouma. Vinoumat kehittyvät koko ihmisen elinkaaren aikana alkaen varhaislapsuudesta. Ne syntyvät suorien ja epäsuorien viestien kautta ympäristöstä, kulttuurista ja sosiaalisista vuorovaikutuksista, mikä muokkaa alitajuntaisesti ihmisen tapaa havainnoida ja tulkita maailmaa. Samaan tapaan myös tekoälyjärjestelmät oppivat vinoumia, sillä ne koulutetaan ihmisten keräämällä datalla, joka heijastaa ihmisten ennakkoasenteita ja stereotyyppejä. Tämän vuoksi tekoälymallit voivat vahvistaa olemassa olevia yhteiskunnallisia epäoikeudenmukaisuuksia, ellei niiden kehitysvaiheessa kiinnitetä tietoista huomiota vinoumien tunnistamiseen ja lieventämiseen.

On tehty monia tutkimuksia, joissa on osoitettu tekoälyjärjestelmien epäoikeudenmukaisuus. Tekoälyjärjestelmien vinoumat ovatkin laaja ongelma, joka vaikuttaa epätasa-arvoisesti koko yhteiskuntaan, etenkin vähemmistöryhmiin.

Esimerkiksi Yhdysvalloissa rikoksen uusiutumisen tunnistamiseen käytetty COMPAS-järjestelmä luokitteli Afrikan amerikkalaiset todennäköisemmäksi tekemään rikoksen uudestaan, vaikka heillä ei olisi ollut taustalla aiempaa tuomiota, ja vastaavia tuloksia on saatu myös muista järjestelmistä [22]. Myös terveydenhuollossa apuvälineinä käytetyistä tekoälyä hyödyntävistä järjestelmistä on löytynyt vinoutumia. Yhdysvaltain terveydenhuollossa käytettävän tekoälyjärjestelmän algoritmi perustaa hoitoon pääsyn luokittelun ennustamalla sairauden sijasta terveyskustannuksia. Yleisesti yhteiskunnassa käytetään vähemmän rahaa tummaihoisten henkilöiden terveydenhoitoon, jolloin algoritmi on arviointihetkellä virheellisesti arvioinut kustannusten perusteella, että tummaihoiset potilaat ovat terveempiä, kuin yhtä sairaat valkoiset potilaat. Tämä on johtanut siihen, että tummaihoisen henkilön hoitoon pääsy on evätty, tai hän on saanut huonompaa hoitoa. Kun algoritmin logiikasta poistettaisiin kustannusperusteisen päätöksenteko, se vähentäisi tätä vinoumaa. [23.]

Tekoälyjärjestelmän vinoumat vaihtelevat riippuen siitä, missä niiden kehittämiseen käytetty data on kerätty ja tekoälymalli opetettu, sekä siitä, kuka on kehittänyt päätöksiä tekevät algoritmit ja miten. Datavinoumat johtuvat datan puutteellisesta laadusta. Datan laatu on heikentynyttä muun muassa silloin, kun se sisältää virheitä, siitä puuttuu, tai siinä on liikaa tietoja, dataa on kerätty hyvin suppealta alueelta ja se sisältää vinoutuneita tai vinoumiin johtavia tietoja. Vinoumaan johtavat tiedot voivat olla esimerkiksi sukupuoli tai postinumero. Algoritmien tuottamat vinoumat johtuvat usein siitä, että niiden päätöksen teon kriteerit ovat vinoutuneita, tai niiden toimintalogiikka perustuu oletetuille vinoumille. Kehittäjä- tai käyttäjävinoumat johtuvat suorasta ihmiskontaktista järjestelmää kehittäessä. Henkilö voi tiedostaen sekä tiedostamattaan opettaa mallia vinoutuneella testidatalla, tai liittää järjestelmään omia vinoumiaan. [24.]

Tekoälyjärjestelmien vinoumien tunnistaminen ja testaus on haasteellista useasta syystä. Koneoppimismallit ja neuroverkot ovat usein monimutkaisia ja niiden toiminta tapahtuu usein niin sanotusti konepellin alla, jolloin mallin toimintalogiikkaa tai päättelyä ei voida nähdä tai selvittää vaihe vaiheelta. On tutkittu, että mallin läpinäkyvyys ja selitettävyys vähentää myös vinoumien syntyä.

Vinouma saattaa olla samanaikaisesti monessa ominaisuudessa, kuten datassa ja algoritmin rakenteessa. Myös mallin käyttötilanne ja mallin itseoppivaisuus voi vaikuttaa vinouman syntyyn, vaikka alkuperäisessä versiossa vinoumaa ei olisi ollut. Tämä aiheuttaa sen, että vinouman juurisyyn löytyminen on usein haasteellista.

Vinouman tunnistaminen vaatii myös syvää ammattitaitoa testaajalta. Mallin toimintaa arvioidessa testaajalla on tärkeää olla tarkka tieto koulutusdatasta. Usein tämä ei ole mahdollista, sillä data voi olla suljettua, tai peräisin kolmannen osapuolen lähteistä, jolloin sen rakenteen ja mahdollisten vinoumien arviointi muodostuu haastavaksi. Vinoumien tunnistamiseen on kehitetty erilaisia testitapoja ja -algoritmeja. Usein vinoumaa ei voida havaita yksittäisen testitapauksen avulla, vaan luotettavien tulosten saaminen vaatii kattavan testijoukon, jotta saadaan näkyviin, suosiiko malli toistuvasti jotakin tiettyä ryhmää. Datan käsittely ja tulosten tilastollinen tulkinta vaatii teknistä ammattitaitoa ja omien ja testitiimin vinoumien tunnistamista, jotta itseään koskemattomat vinoumat eivät jää sokeiksi pisteiksi vinoumia etsiessä. [24.]

5.1.3 Datan puute tai laatu

Tekoälyjärjestelmien testauksessa käytettävän datan puute tai heikko laatu muodostaa merkittävän haasteen mallien luotettavalle arvioinnille. Virheellinen data vaikuttaa mallin opetustuloksiin ja voi aiheuttaa malliin rakenteellisia haavoittuvuuksia. Järjestelmissä, joissa mallin käyttö perustuu nopeatahtiseen, suurivolyymiseen ja monimuotoiseen dataan, kuten kuviin, testidatan hankinta voi olla erityisen vaikeaa, mikä aiheuttaa haasteita luoda edustavaa testiaineistoa, joka kuvaisi todellista tuotantoympäristöä.

Kattavaa ja laadukasta testidataa voi olla haasteellista kerätä ja tuottaa tarvittavan nopeasti, jolloin testidata ei vastaa koulutusdatan laatua. Tämä voi johtaa siihen, että mallien testaamisessa käytetty data on heikkolaatuista, eikä testauksesta saada luotettavasti tuloksia mallin totuudenmukaisesta käyttäytymisestä. Heikko datan laatu testauksessa käytettynä altistaa myös itseoppivia malleja

oppimaan ei haluttuja toimintatapoja. Tekoälymalleissa tarvittavat datamäärät ovat suuria, ja ne voivat olla peräisin monista eri lähteistä. Datan käsittely ja prosessointi on aikaa vievää, ja vaatii ymmärtämistä sovellusalueista, datasta ja sen ominaisuuksista sekä datankäsittelyn tekniikoista. Datan käsittely sisältää paljon manuaalista työtä, sillä toiminnallisuutta ei voi automatisoida kokonaan. Tämä kasvattaa kehityskustannuksia, ja etenkin testauksen yhteydessä tapahtuva datan käsittely altistaa inhimillisille virheille, sillä vaikka mallien rakentamisen yhteydessä datankäsittelyn tekee datankäsittelyyn erikoistunut ammattilainen, testivaiheessa datankäsittelyn tekevät usein testaajat ja usein ilman kattavaa koulutusta. [19.]

Datalle tehdään esiprosessointi, jossa data siivotaan käytettävään ja yhteneväisesti tallennettavaan muotoon. Tyypillisesti esiprosessointivaihe sisältää monia poisto-operaatioita, joissa poistetaan virheelliset, määrittelemättömät ja toisteiset arvot sekä poikkeavuudet, jotka vaikuttavat olevan kaukana viitearvoista. Tämän lisäksi epäjohdonmukaisten numeroasteikkojen ja useiden tietomuotojen välttämiseksi muuttujien yksiköt yhtenäistetään. Yleisiä datan muotoilutoimenpiteitä ovat esimerkiksi normalisointi, jossa keskiarvo on nolla ja hajonta yksi, sekä skaalaus, jossa arvot muutetaan välille 0-1. Mikäli datan esiprosessointivaiheet tehdään puutteellisesti, voi dataan jäädä turhia tai vanhentuneita tietueita. Jo yksittäisten heikkolaatuisten ominaisuuksien mukanaolo voi lisätä datan kohinaa ja vähentää mallin suorituskykyä etenkin edistyneimmissä tarkemmissa tekoälymalleissa. [25.]

Dataa käsitellään usein myös dataputkien (data pipelines) avulla. Dataputki on automatisoitu prosessi sisältäen joukon yhdistettyjä tietojenkäsittelytoimintoja, jotka siirtävät tietoja järjestelmästä toiseen. Dataputket voivat sisältää erilaisia vaiheita, kuten datan keräämistä ja sen puhdistamista, muuntamista ja tallentamista. Nämä putket räätälöidään liiketoiminnan tarpeiden mukaan ja niitä voi olla samassa järjestelmässä useita. [26.] Varsinkin suurien ja epästruktuuristen datamäärien käsittelyssä dataputkia voi olla useita eri tarpeisiin kehitettyjä, jolloin niistä voi koostua ”putkiviidakko”-ongelma, jossa putkia on liikaa, eikä kaikkien tarkoitusta tai toiminnallisuutta tunneta, ja samat toiminnallisuudet voivat

toistua useissa putkissa. Tämmöinen toteutus on herkkä koodivirheille ja virheiden jäljittäminen on haasteellista. Myös prototyyppisesti koodattujen putkien heikko laatu ja niiden koodista siivoamatta jääminen voi aiheuttaa mallille muutoksia ja epätoivottua käytöstä reaalimaailmassa. [27.] Testaajat testaavat näitä monimutkaisia dataputkia sekä rakentavat niitä myös itse testidatan käsittelyä varten. Monimutkaisten putkien verkon testaus on aikaa vievää ja vaatii ymmärrystä niiden toiminnasta. Mikäli dataputkista ei ole saatu tehty skaalattuja ja suorituskykyisiä, jossa tietueen käsittely tapahtuisi nopeasti, niiden testaaminen suurilla testidatoilla on hidasta. [28.]

5.2 Tekoälyjärjestelmien vaatimukset

Tekoälylle tai sen käytölle järjestelmissä on asetettu vaatimuksia ja säädöksiä. Säädökset vaihtelevat runsaasti eri maanosien välillä, joten kaikilta osin täsmääviä yhteisiä globaaleja säännöstelyjä ei toistaiseksi ole olemassa. Euroopan unioni (EU) on laatinut tekoälyasetuksen nimeltä AI Act, jolla pyritään varmistamaan, että tekoälyä on turvallista ja eettistä käyttää samalla suojellen perusoikeuksia ja tukien innovaatiota. AI Act astui voimaan elokuussa 2024, ja vuoden 2027 puolessa välissä yritysten tulisi täyttää kaikki vaatimukset tekoälyn käytön osalta. AI Act vaikuttaa myös EU:n ulkopuolella, sillä myös ulkomaiset toimijat ovat asetuksen piirissä, jos ne tuovat tekoälyjärjestelmiä EU:n markkinoille tai niiden tuotoksia käytetään EU:ssa [29].

EU:n tekoälyasetus [30] luokittelee tekoälyjärjestelmät neljään riskitasoon:

1. Hyväksymätön riski: Tämän riskitason sovelluksia ja niiden toiminnallisuuksia, kuten sosiaalinen luokittelu ja biometrinen tunnistus julkisilla paikoilla, on kiellettyä käyttää.

2. Korkea riski: Korkean riskin järjestelmät sisältävät riskejä vaikuttaa turvallisuuteen, terveyteen tai perusoikeuksiin negatiivisesti. Korkean riskin tason käyttöalueeseen on tällä hetkellä luokiteltu esimerkiksi terveydenhuolto.

3. Vähäinen riski: Nämä ovat sovelluksia, joissa vaaditaan läpinäkyvyyttä, kuten käyttäjälle selkeä tieto siitä, että ovat tekoälyn kanssa tekemisessä. Esimerkkisovellus voisi olla chatbot.

4. Ei riskiä tai minimaalinen riski: Tähän tasoon kuuluvat useimmat tekoälyjärjestelmät, jotka jo noudattavat lainsäädäntöä, eikä niihin liity erillisiä lisävaatimuksia. Esimerkkijärjestelmänä roskapostin suodattimet.

Testauksen näkökulmasta hyväksymättömän riskin sovelluksille ei tarvitse edes aloittaa testausprosessia, sillä sovellukset ovat yksinkertaisesti kiellettyjä kehittää. Korkean riskitason sovelluksille on laajat testivaatimukset, joihin kuuluu esimerkiksi

- toiminnallinen testaus, jossa varmistetaan, että järjestelmä toimii asianmukaisesti, esimerkiksi lääketieteessä tekoäly antaa oikeat diagnoosit tai diagnoosivaihtoehdot

- tietoturvatestaus, jossa simuloidaan hyökkäyksiä esimerkiksi penetraatiotestauksella, ja tarkistetaan, ettei kuulumaton taho saa pääsyä sensitiivisiin tietoihin

- eettinen testaus, jossa selvitetään, noudattaako tekoäly eettisiä periaatteita ja onko päätöksen teko vinoutunutta

- käytettävyystestaus, jossa varmistetaan, että tekoäly on ihmisen valvottavissa ja tarvittaessa keskeytettävissä.

Vähäisen ja minimaalisen riskin sovellusten testaus on pääosin perinteistä testausta ja keskittyy muun muassa toiminnalliseen testaukseen, läpinäkyvyyteen ja tietoturvaan. Vaatimusten näkökulmasta testauksessa varmistetaan, että käyttäjälle ilmoitetaan selkeästi tekoälyn käytöstä ja käyttäjän tietoja käsitellään turvallisesti.

6 Tekoälyjärjestelmien testauksen tasot

Ohjelmistokehityksen testaustasot kuvaavat ohjelmistokehityksen vaiheita testauksen aikana ja ne sisältävät erityyppisiä testausmenetelmiä. Tässä luvussa käsitellään pääosin tekoälyjärjestelmien testauksen tasoja, mutta myös muutamia tasoilla suoritettavia testausmenetelmiä käydään läpi suhteellisen pintapuolisesti.

Tekoälyjärjestelmien voidaan ajatella olevan hybridijärjestelmiä, jotka sisältävät sekä perinteisiä ohjelmistokomponentteja että tekoälykomponentteja. Tekoälyjärjestelmät noudattavat osittain monia samoja perinteisen ohjelmistotestauksen tasoja, kuten aiemmin tässä työssä mainittuja yksikkö-, integraatio-, järjestelmä- ja hyväksymistestausta, mutta lisätasoja tekoälyjärjestelmissä ovat testidatan testaus ja mallin testaus. Mallin testaus on laaja taso, joka sisältää monia alatasoja, joista tässä luvussa niistä käsitellään turvallisuustestausta ja vinoumien testausta.

6.1 Perinteiset testaustasot tekoälyjärjestelmissä

Perinteisissä ohjelmistoissa yksikkötestaus keskittyy yksittäisten ohjelmistofunktioiden, metodien tai komponenttien testaamiseen itsenäisinä toimijoina. Tekoälypohjaisissa sovelluksissa ei-tekoälykomponenttien testauksen lisäksi yksikkötestaus sisältää ensisijaisesti datan esikäsittelytoimintojen, ominaisuuksien (feature) poiminnan (ekstraktio) ja yksittäisten mallikomponenttien testaamisen [8]. Datan voidaan ajatella olevan tekoälyjärjestelmän komponentti. Tässä vaiheessa harvemmin löydetään varsinaisia virheitä, ja tämän vaiheen tavoite on parantaa tuotettavan mallin laatua. Yksikkötestauksessa voidaan asettaa mallin suorituskyvylle hyväksymiskriteerejä, kuten tarkkuus ja herkkyys, joita vastaan mallia testataan. [19.]

- Tekoälymallit sisältävät usein esikäsittelyputkia, joissa dataa puhdistetaan, muunnetaan ja valmistellaan koulutusta varten. Jokainen tämän

putken toiminto on testattava sen varmistamiseksi, että se käsittelee erilaiset tiedostomuodot ja reunatapaukset oikein.

- Ominaisuuksien poimintamekanismit on validoitava, jotta voidaan varmistaa, että ne laskevat arvot oikein raakadatan perusteella.

- Jos tekoälymalli on modulaarinen, on tarpeellista testata yksittäisiä alimalleja tai kerroksia (esim. neuroverkoissa). Testikattavuus neuroverkon testauksessa voidaan määritellä verkkojen testauksen kattavuudella.

Integraatiotestauksessa tavoitteena on varmistaa, että eri ohjelmistomoduulit toimivat oikein yhdessä. Tekoälykomponenttia voidaan testata sen olevan joko kiinteänä osana järjestelmää, tai tarjottuna esimerkiksi verkkopalveluna, jolloin komponentti on käytössä kutsuttaessa. [19.] Tekoälypohjaisten sovellusten testauksessa tässä vaiheessa keskitytään myös datan käsittelyketjuun, malliin ja API-rajapintoihin [8].

- Tarkistetaan, että mallit integroituvat oikein ulkoisiin rajapintoihin, tietokantoihin tai muihin järjestelmiin, jotka toimittavat reaaliaikaista dataa.

- Varmistetaan, että syötteet siirtyvät oikeassa muodossa datan käsittelyvaiheesta tekoälymalliin ja että mallin ennusteet käsitellään ja näytetään oikein käyttöliittymässä.

Järjestelmätestauksessa varmistetaan koko järjestelmän yhteensopivuus ja toiminta. Tekoälypohjaisissa sovelluksissa tämä vaihe keskittyy end-to-end-tyyppiseen validointiin, jossa varmistetaan, että malli tuottaa käyttöympäristössä luotettavia ja tarkkoja tuloksia [16].

- Testataan mallin käyttäytymistä erilaisissa tilanteissa, mukaan lukien odottamattomat syötteet ja reunatapaukset.

- Tarkistetaan tekoälyn generoimien tulosten luotettavuus todellisten syötteiden perusteella.

- Lisäksi testataan järjestelmän ei-toiminnallisia ominaisuuksia, kuten tietoturvaa ja mallin ennusteaikojen suorituskykyä. [19.]

Hyväksymistestauksen tavoitteena on varmistaa, että ohjelmisto täyttää liiketoimintavaatimukset ennen sovelluksen todellista käyttöönottoa. Tekoälypohjaisissa sovelluksissa tässä vaiheessa arvioidaan myös mallin tarkkuus, puolueettomuus, vinoumat ja käyttäjäkokemus.

- Asiantuntijavalidointi varmistaa, että tekoälyn tuottamat tulokset vastaavat alan odotuksia (esimerkiksi terveydenhuollossa tai rahoitusalalla).

- Käyttäjätestaus arvioi, tarjoavatko tekoälypohjaiset ominaisuudet merkityksellisiä ja hyödyllisiä tuloksia loppukäyttäjille.

- Tarkistetaan säädöstenmukaisuus, erityisesti sovelluksissa, joissa mallin selitettävyys ja oikeudenmukaisuus ovat lakisääteisiä vaatimuksia.

Tekoälyjärjestelmissä järjestelmän kehittyessä ja käyttöympäristön muuttuessa tulee suorittaa säännöllistä ylläpitotestausta. Ylläpitotestauksessa varmistetaan, että tekoälyjärjestelmä täyttää alkuperäiset hyväksymiskriteerit edelleen mallin kehittymisen jälkeenkin.

Koska tekoälyjärjestelmät ovat monimutkaisempia, ne tarvitsevat niiden ainutlaatuisen luonteen vuoksi lisäksi muita testauksen tasoja validoidakseen muun muassa mallien suorituskyvyn ja reiluuden. Tekoälyjärjestelmiä on useita erilaisia, joten kaikki testauksen tasot tai tyypit eivät ole sovellettavissa kaikkien eri tekoälyjärjestelmien testaukseen. Datakeskeinen testaus ja mallin validointitestaus, johon kuuluvat eettisyys ja turvallisuustestaus ovat mukana useissa tekoälyjärjestelmien testausprosesseissa.

6.2 Datakeskeinen testaus

Data on tekoälyn perusta. Ilman suuria datamääriä tekoälyä ja siihen liittyvää koneoppimista olisi mahdotonta tehdä. Datan testaus on kriittinen vaihe

testauksessa, sillä datan laatu, reiluus ja tarkkuus ovat suoraan verrannollisia tekoälyä käyttävän ohjelmiston laatuun. Heikkolaatuinen data voi johtaa vinoutuneisiin algoritmeihin ja epäluotettaviin lopputuloksiin. Datan laadun testaamisella pyritään varmistamaan, että tekoälymallin toiminnat perustuvat luotettavaan tietoon, eikä se tee päätöksiä epätarkkojen syötteiden pohjalta. [31.]

Datan laadun testauksessa on tavoitteena varmistaa, että tekoälymallin koulutuksessa käytettävä data on virheetöntä, ajantasaista, tarkkaa ja yhtenäistä. Virheettömällä datalla tarkoitetaan sitä, että siitä ei löydy kirjoitusvirheitä, puuttuvia tietoja tai rakenteellisia ongelmia. Jotta tekoälymalli tekisi päätöksiä oikean ja ajankohtaisen tiedon pohjalta, eivätkä päätökset perustuisi vanhentunteisiin tietoihin, datan täytyy olla ajantasaista. Tarkkuus takaa sen, että data antaa täsmällistä ja luotettavaa informaatiota päätöksen tueksi ja uniikkius varmistaa, ettei tietoja ole toistettu tai esiinny päällekkäisyyksiä. Kokonaisdatan eheyden ja saatavuuden varmistaa yhtenäinen data, eli data on tallennettu keskitetysti, eikä sitä ole hajautettu useisiin eri järjestelmiin. [31.]

Testisyötteiden testaus sisältää eri menetelmiä, kuten tilastollinen analyysi, jossa syötedatasta voidaan etsiä esimerkiksi vinoumia, ja eksploratiivinen dataanalyysi (EDA), jolla pyritään hahmottamaan datan rakennetta, jakaumia ja poikkeamia. Datan käsittelyssä käytetään useita erilaisia toisiinsa yhteydessä olevia dataputkia. Syötedatan käsittelyyn voi liittyä prototyyppiputki ja tuotantoputki. Prototyyppiputkea käytetään mallin koulutuksen aikana ja tuotantoputkea tuottamaan reaaliaikaista dataa ennustamiseen. Dataputkille on tärkeää tehdä sekä yksikkö- että integraatiotestejä, jotta data pysyy ehjänä koko putken, tai putkien, läpimenoajan. Yksikkötestauksessa testataan yksittäisen tietoputken eri osien, kuten validointi- ja muunnoskomponenttien, toimivuus. Integraatiotestauksessa testataan yksittäisen putken kokonaisuutta sekä useamman putken yhteensopivuutta. [28.]

Tällä testaustasolla voidaan käyttää myös synteettisen datan luomista, eli prosessia, jossa tietojoukolle luodaan tietoprofiili. Tietoprofiilin avulla organisaatiot voivat verrata, kuinka heidän oma todellinen datansa vertautuu tämän jo

validointisäännöillä määritetyn tietojoukon tietolähteisiin ja luokitella, onko heidän datansa laatumääritysten mukainen. [32.]

6.3 Tekoälymallin testaus

Tekoälymallin testaus on keskeinen osa tekoälyjärjestelmien testausprosessia, sillä se varmistaa, että malli toimii tarkoitetulla tavalla ja tuottaa luotettavia, tarkkoja ja johdonmukaisia tuloksia. Mallin validoinnissa arvioidaan sen suorituskykyä eri näkökulmista, kuten tarkkuuden, yleistettävyyden, luotettavuuden ja ennustettavuuden osalta. Ilman huolellista validointia tekoälymalli saattaa toimia hyvin testidatalla, mutta epäonnistua reaalimaailman tilanteissa, joissa tiedon rakenne ja sisältö voivat vaihdella merkittävästi

Mallin validoinnissa käytetään erilaisia testausmenetelmiä, kuten ristiinvalidointia, hajautettua testidataa ja A/B-testausta. Ristiinvalidointi auttaa varmistamaan, että malli ei ole liian ylisovitettua tiettyyn datasettiin, vaan sopeutuu hyvin aiemmin näkemättömiin tapauksiin. Ristiinvalidoinnissa data jaetaan osiin eri tavoin ja mallia koulutetaan ja testataan vuorotellen eri osilla. Hajautettu testidata koostuu erityyppisistä ja eri lähteistä tulevista tiedoista, joiden avulla voidaan varmistaa, ettei malli ole liian sidottu tiettyyn datarajaan tai skenaarioon. A/B-testauksessa vertaillaan eri malliversioiden suorituskykyä käytännön tilanteissa, jotta voidaan määrittää, mikä versio tuottaa parhaat lopputulokset. [28.]

Mallin validointia täydennetään erilaisilla testioraakkeleilla. Toleranssioraakkelia käytetään, kun täsmälliseen arvoon osumisen sijasta riittää, että tulos osuu määritetylle hyväksymisalueelle. Tässä kuitenkin tarkastellaan yksittäistä tulosta. Tekoälyjärjestelmissä mallin suorituskykyä arvioidaan aggregaattimittareilla, kuten tarkkuus (accuracy), herkkyys (recall) ja niistä muodostuvalla F1-arvolla. Tässä yksittäiset virheet tuloksissa voivat olla hyväksyttäviä, kunhan tulosten kokonaislaatu pysyy riittävän korkealla läpi koko testiaineiston. Tätä ei kuitenkaan vakiintuneesti kutsuta ”tilastolliseksi oraakkeliksi”, vaan tämä kuvaa useamman oraakkelityypin toimintatapaa perustaessaan arviointinsa mittaripohjaiseen analyysiin useiden suoritusten tuloksista. Metamorfinen oraakkeli

perustuu syötteen ja mallin tuottaman tuloksen välisiin odotettuihin suhteisiin. Syötettä muutetaan hallitusti ja tarkastellaan, muuttuuko tulos loogisesti ja odotetulla tavalla. [25.] Esimerkiksi kuvantunnistusohjelmassa mallin tulisi tunnistaa sama kuva, vaikka kuvan kirkkautta säädettäisiin, ja kielentunnistussovelluksessa alkuperäisellä lauseella ja lauseella, jossa on muutettu välimerkkejä, pitäisi olla sama tunnistettu kieli. Metamorfinen testaus on hyödyllistä mustan laatikon testauksessa (black box testing), joissa mallin sisäistä logiikkaa ei voida suoraan tutkia tai nähdä. Tämän avulla voidaan havaita epäjohdonmukaisuuksia ja robustisuuspuutteita mallin toiminnassa, vaikka ei tunnettaisi täsmällistä oikeaa tulosta.

Tekoälymallin validointia on tehtävä jatkuvasti mallin elinkaaren aikana. Malli saattaa kohdata uusia haasteita, kuten datamuutoksia, konseptin ajautumista (concept drift) ja muuttuvia käyttäjävaatimuksia, jotka voivat heikentää sen suorituskykyä ajan myötä. Tästä syystä mallin testaus ja validointi ovat jatkuvia ja iteratiivisia prosesseja, joiden avulla varmistetaan, että tekoäly pystyy mukautumaan muuttuviin olosuhteisiin ja säilyttämään suorituskykynsä.

6.3.1 Turvallisuustestaus

Tekoälyjärjestelmät sisältävät paljon dataa ja siten myös sensitiivistä tietoa. Niin kuin kaikkia järjestelmiä, myös tekoälyjärjestelmää tulee testata turvallisuuden näkökulmasta. Tietoturvatestauksen tavoitteena ovat haavoittuvuuksien löytäminen ja varmistaa, että tekoälyjärjestelmä on suojattu hyökkäyksiltä ja luvattomalta käytöltä. Tähän sisältyy turvallisen viestinnän, tietosuojan ja kyberhyökkäyskestävyyden testaus. [33.] Tietoturvaa voidaan testata esimerkiksi adversiaalitestauksella ja penetraatiotestauksella.

Adversiaalitestauksen tavoitteena on arvioida ja testata, miten tekoälyjärjestelmä reagoi ja kestää hyökkäyksiä, joissa järjestelmään syötetään tarkoituksellisesti manipuloitua tai haitallista dataa. Tällä voidaan tunnistaa tekoälymallien mahdolliset haavoittuvuudet ja täten parantaa järjestelmän kykyä kestää epäluotettavia syötteitä. Penetraatiotestaus tarkoittaa eettistä kyberhyökkäyksen

tekemistä. Siinä simuloidaan todellisia kyberhyökkäyksiä tavoitteena tunnistaa tekoälyjärjestelmän tietoturva-aukot. Tämä menetelmä auttaa tunnistamaan järjestelmän heikot kohdat ja varmistamaan, että järjestelmä kestää yleisimmät hyökkäystavat, kuten luvattoman pääsyn, injektiohyökkäykset ja tietovuodot. [34.]

Tekoälyjärjestelmän turvallisuustestauksen tekevät usein kokeneet tietoturvan ammattilaiset, joilla on kokemusta myös tekoälyjärjestelmistä. Turvallisuustestauksen toteuttamiseksi on hyvä ymmärtää tekoälyn peruskonsepteja, kuten datan esikäsittelyä, mallien koulutusta ja koneoppimisalgoritmeja. Näiden ymmärtäminen auttaa tunnistamaan mahdolliset hyökkäyspinnat ja turvallisuusriskit ja siten kehittää testauksen optimaaliseksi.

Testaamisessa tulee myös huomioida datan käsittelyyn liittyvät turvallisuusvaatimukset, kuten tietosuoja, tiedon salaaminen ja pääsynhallinta. Tämän lisäksi on tärkeää ottaa huomioon kolmannen osapuolen komponenttien turvallisuus, kuten erilaiset kirjastot ja ohjelmistokehykset (frameworks), ja varmistaa, että ne ovat ajantasaisia eivätkä sisällä tunnettuja haavoittuvuuksia.

6.3.2 Vinouma- ja oikeudenmukaisuustestaus

Vinouma- ja oikeudenmukaisuustestaus tekoälyjärjestelmissä pyrkii tunnistamaan, analysoimaan ja vähentämään vinoumia. Tämä on merkittävä testauksen osa-alue, jolla pyritään varmistamaan, ettei tekoäly aiheuta tai ylläpidä epäoikeudenmukaisuutta tai syrjintää. Tekoälyjärjestelmissä datasta aiheutuvien vinoumien testaus voidaan jakaa kahteen vaiheeseen – opetusdatassa olevan vinouman tunnistaminen ja poistaminen, kuten etninen tausta, sukupuoli ja ikä, sekä testaamalla järjestelmä vinoumista vapaalla testidatalla [35].

Vinoumien varalta voidaan käyttää myös muita testausmenetelmiä, kuten vinoumien tunnistamista tilastollisilla analyyseillä sekä selittäviä algoritmeja, jotka avaavat mallin päätöksentekoprosessia ymmärrettävämmäksi ja läpinäkyvämmäksi. Näihin on julkaistu avoimia työkaluja ja algoritmeja kuten IBM:n AI

Fairness 360 toolkit, joka pyrkii edistämään koneoppimismallien vinoumien tunnistamista. Käytännön testaus erilaisilla käyttäjäryhmillä auttaa tunnistamaan mahdollisia syrjiviä käytäntöjä, joita pelkkä tilastollinen analyysi ei välttämättä paljasta. Sen vuoksi vinoumien testaus sisältää myös ihmisten osallistumista, jossa järjestelmien päätöksiä arvioidaan sekä asiantuntijoiden että käyttäjien näkökulmasta, jotta varmistetaan järjestelmän eettisyys ja hyväksyttävyys. Lisäksi hyödynnetään turvallisuustestauksessa mainittua adversiaalitestausta, jossa malli altistetaan tilanteille, joissa tarkoituksellisesti haastetaan mallia vääristetyllä tai poikkeavalla datalla. Tämä voi auttaa havaitsemaan eettisiä puutteita ja riskejä.

Aina järjestelmästä ei pystytä poistamaan kaikkia vinoumia johtuen järjestelmän monimutkaisuudesta ja sen itseoppivaisuudesta, ja on hyväksyttävä, että julkaistu järjestelmä sisältää joko implisiittisiä tai eksplisiittisiä vinoumia. Tällöin olisi eettistä ja läpinäkyvyyttä lisäävää julkaista samalla myös mallin kehityksessä käytetty opetusdata. [19.] Vuosien kuluessa myös käsitykset vinoumista ja syrjivistä tuloksista voivat muuttua ja laajentua. Näiden vuoksi testausta vinoumien varalta tulee jatkaa säännöllisesti myös järjestelmän julkaisun jälkeen.

7 Yhteenveto

Tässä työssä tarkasteltiin ohjelmistotestauksen periaatteita sekä tekoälyjärjestelmien testaamisen erityispiirteitä. Työssä todettiin, että tekoälyjärjestelmien testaamisen suurimmat haasteet ja erot perinteisten järjestelmien testaukseen jakautuvat testidataan liittyviin haasteisiin sekä mallien toiminnallisiin ja eettisiin haasteisiin. Näitten erityispiirteitten takia perinteiset testausmenetelmät eivät yksin riitä takaamaan tekoälyjärjestelmien toimivuutta ja luotettavuutta.

Testidatan haasteet liittyvät ensisijaisesti datan laatuun, kattavuuteen ja edustavuuteen. Tekoälymallin luotettavuus ja suorituskyky ovat suoraan riippuvaisia koulutus- ja testidatasta, jolloin datan virheet, puutteet tai harhat voivat johtaa epätarkkoihin tai harhaanjohtaviin testituloksiin. Ratkaisuksi tähän on esitetty datakeskeistä testausta, jossa datan eheys ja laatu varmistetaan kattavalla

validoinnilla ennen mallin arviointia. Dataprosessien laadun varmistamiseksi myös dataputkien testaus muodostaa olennaisen osan laadunvarmistusta.

Mallien toiminnalliset ja eettiset haasteet liittyvät tekoälyjärjestelmien epädeterministiseen käyttäytymiseen, vinoumiin sekä turvallisuuteen ja läpinäkyvyyteen liittyviin ongelmiin. Epädeterministisyyden vuoksi perinteinen, yksiselitteisiin odotettuihin tuloksiin perustuva testaus ei ole sovellettavissa sellaisenaan. Tähän haasteeseen vastaavat toleranssipohjaiset arviointimenetelmät, joissa tulosten hyväksyttävyys määritellään arvoalueina yksittäisten tarkkojen arvojen sijaan, sekä metamorfinen testaus, joka arvioi mallin loogista johdonmukaisuutta muokkaamalla mallille annettua syötettä hallitusti.

Mallien yleistettävyyden ja luotettavuuden arvioinnissa hyödynnetään erityisesti ristiinvalidointia, jossa mallin arviointi tapahtuu erilaisilla datan osajoukoilla ylisovittamisen tunnistamiseksi. Ristiinvalidoinnin haasteena on kuitenkin sen vaatima infrastruktuuri: menetelmä edellyttää usein rinnakkaisia järjestelmiä tai kopioita mallista, mikä voi olla käytännössä hankalaa tai kallista toteuttaa.

Tekoälyjärjestelmien kohdalla erityisen kriittinen alue on vinoumien tunnistaminen ja eettisyyden varmistaminen. Järjestelmän vinoumat voivat johtaa epäreiluun ja syrjivään päätöksentekoon. Vinoumien tunnistamisessa käytetään tilastollisia analyysejä ja selittäviä algoritmeja, jotka auttavat ymmärtämään mallin päätösten perusteita. Lisäksi adversiaalitestauksen ja käyttäjäryhmillä tehtävien käytännön testien avulla voidaan tunnistaa ongelmia, jotka eivät välttämättä ilmene pelkässä tilastollisessa arvioinnissa.

Toisin kuin perinteisten järjestelmien tapauksessa, tekoälyjärjestelmien testaus ei pääty järjestelmän käyttöönottoon. Mallien suorituskyky voi heiketä ajan myötä esimerkiksi konseptin ajautumisen seurauksena, mikä korostaa jatkuvan ylläpitotestauksen ja hyväksymiskriteerien säännöllisen uudelleenarvioinnin tarvetta.

Työssä tunnistettiin myös, että yksi alan keskeisistä haasteista on testausmenetelmien ja arviointikäytäntöjen hajanaisuus. Toistaiseksi alalta puuttuvat

yhtenäiset ja vakiintuneet testausstandardit, mikä vaikeuttaa eri tekoälyjärjestelmien vertailua ja luotettavan laadun arviointia. Euroopan unionin uusi tekoälyasetus (AI Act) ei suoraan määrittele teknisiä testausmenetelmiä, mutta luo sääntelykehikon, joka vahvistaa tekoälyjärjestelmien läpinäkyvyyden, turvallisuuden ja oikeudenmukaisuuden vaatimuksia. Tämän seurauksena organisaatioiden täytyy kyetä osoittamaan järjestelmiensä vaatimustenmukaisuus myös testauksen avulla, mikä kannustaa testauskäytäntöjen kehittämiseen ja yhtenäistämiseen.

Tekoälyjärjestelmien testaus on monitahoinen prosessi, joka edellyttää perinteisten testausmenetelmien lisäksi erityisosaamista tekoälyn erityispiirteistä. Testausprosessin onnistuminen vaatii teknistä, tilastollista ja eettistä osaamista, sekä selkeitä käytännön menetelmiä. Tulevaisuudessa tarvitaan erityisesti lisää tutkimusta ja alan yhteistyötä testausmenetelmien vakiinnuttamiseksi, jotta tekoälyjärjestelmien laatu, luotettavuus ja yhteiskunnallinen hyväksyttävyys voidaan varmistaa tehokkaasti ja läpinäkyvästi. Ohjelmistotestaus ei voi olla enää vain ohjelman teknisen toiminnan varmistamista, vaan sen on katettava myös tekoälyn päätöksenteon oikeudenmukaisuus, läpinäkyvyys ja turvallisuus.

Lähteet

1 Graham Dorothy; van Veenendaal Erik; Evans Isabel & Black Rex. Foundations of Software Testing. ISTQB Certification. Luettu 25.3.2025.

2 McGuire Mike. 2022. What is the cost of poor software quality in the U.S.? Verkkoaineisto. <https://www.blackduck.com/resources/analyst-reports/cost-poor-quality-software.html> Luettu 15.2.2025.

3 Leveson Nancy. Medical Devices: The Therac-25*. Verkkoaineisto. <http://sunnyday.mit.edu/papers/therac.pdf> Luettu 15.2.2025.

4 Whitney Robert. Testaus vs virheenkorjaus. Mikä on ero? Verkkoaineisto. <https://firmbee.fi/testaus-vs-vianetsinta> Luettu 15.2.2025.

5 Sommerville Ian. Software Engineering. Yhdeksäs painos. <https://engineering.futureuniversity.com/BOOKS%20FOR%20IT/Software-Engineering-9th-Edition-by-Ian-Sommerville.pdf> Luettu 28.2.2025.

6 Beck Kent. Test-driven development : by example. 2002. <https://archive.org/details/est-driven-development-by-example/test-driven-development-by-example/page/n33/mode/2up>Luettu 18.2.2025.

7 Kornaga-Deviniti Katarzyna. 2024. The biggest software testing challenges. Verkkoaineisto. <https://community.atlassian.com/forums/App-Central-articles/The-biggest-software-testing-challenges/ba-p/1149462>Luettu 18.2.2025.

8 AI Software Testing Life Cycle: A Step-by-Step Guide. Verkkoaineisto. Confedo AI. <https://www.linkedin.com/pulse/ai-software-testing-life-cycle-step-by-step-guide-blogo-ai-8bxqf/> 2024. Luettu 22.2.2025.

9 What is Software Testing Life Cycle (STLC)? Phases, Models, Interview Questions, SDLC Difference. Verkkoaineisto. TRY QA. <https://tryqa.com/what-is-software-testing-life-cycle-stlc/>Luettu 22.2.2025.

10 Software Testing Life Cycle (STLC). Verkkoaineisto. T-pointTech. <https://www.tpointtech.com/software-testing-life-cycle#environment-setup> 17.3.2025. Luettu 20.2.2025.

11 Calvello Mara. The 4 Levels of Testing in Software Engineering Explained. Verkkoaineisto. Fellow. <https://fellow.app/blog/engineering/the-levels-of-testing-in-software-engineering-explained/> 8.11.2022. Luettu 22.2.2025.

12 Humble Jez; Farley David. Continuous delivery. <https://pro-web.md/ftp/carti/Continuous-Delivery-Jez%20Humble-David-Farley.pdf> Luettu 22.2.2025.

13 Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Helsinki: Docendo. Luettu 1.3.2025 .

14 Acceptance Testing – Software Testing. Verkkoaineisto. Geeks for geeks. <https://www.geeksforgeeks.org/acceptance-testing-software-testing/> Päivitetty 13.6.2024. Luettu 22.2.2025.

15 Mishra Akanksha. A Comprehensive Review of Artificial Intelligence and Machine Learning : Concepts, Trends, and Applications. <https://www.researchgate.net/publication/384231012_A_Comprehensive_Review_of_Artificial_Intelligence_and_Machine_Learning_Concepts_Trends_and_Applications> 20.9.2024. Luettu 14.3.2025.

16 Bui Thang Duc. AI Model Testing: Crafting Reliable AI Models for Tomorrow. Blogikirjoitus. https://smartdev.com/ai-model-testing-guide/ 10.12.2024. Luettu 14.3.2025.

17 AI and Software Quality: Trends and Executive Insights. Verkkoaineisto. Leapwork. <https://www.leapwork.com/download/ai-and-software-quality-report> 2024. Luettu 13.3.2025.

18 10 Nuances of Testing AI-Based Systems. Blogiteksti. Apriorit. <https://www.apriorit.com/qa-blog/661-qa-nuances-of-testing-ai-based-systems> 20.2.2020. Luettu 13.3.2025.

19 ISO/IEC TR 29119-11:2020. Luettu 15.4.2025.

20 Homès Bernard. 2012 Fundamentals of Software Testing. John Wiley & Sons, Inc. Luettu 15.4.2025.

21 Pham Phu Vinh. The challenges in Testing AI-Based Systems. Blogikirjoitus. Nash Tech. <https://blog.nashtechglobal.com/the-challenges-in-testing-ai-based-systems/> 13.3.2023. Luettu 12.3.2025.

22 Angwin, J.; Larson, J.; Mattu, S.& Kirchner, L. Machine bias. In Ethics of Data and Analytics. 2016. Auerbach Publications: Boca Raton, FL, USA. Luettu 20.3.2025.

23 Obermeyer, Z.; Powers, B.; Vogeli, C. & Mullainathan, S. Dissecting racial bias in an algorithm used to manage the health of populations. 2019. Luettu 20.3.2025.

24 What is AI bias? Verkkoaineisto. SAP. <https://www.sap.com/resources/what-is-ai-bias> 30.10.2024. Luettu 17.4.2025.

25 Houssem Ben Braiek; Foutse Khomh. On Testing Machine Learning. 2018. SWAT Lab., Polytechnique Montr´eal, Canada Programs. Luettu 10.4.2025.

26 Khandavilli Preetipadma. 5 Types of Data Pipelines with Critical Benefits. Nettiartikkeli. Hevo. <https://hevodata.com/learn/types-of-data-pipeline/> Päivitetty 15.10.2024. Luettu 27.4.2025.

27 Hidden Technical Debt in Machine Learning Systems. 2015. D. Sculley; Gary Holt; Daniel Golovin; Eugene Davydov & Todd Phillips. Luettu 27.4.2025.

28 Certified Tester AI Testing. ISTQB. 10.1.2021. Luettu 27.4.2025.

29 Understanding the EU AI Act: Requirements and Next Steps. Nettiartikkeli. ISACA. <https://www.isaca.org/resources/white-papers/2024/understanding-the-eu-ai-act> 18.10.2024. Luettu 20.4.2025.

30 Artificial Intelligence – Questions and Answers. Verkkoaineisto. EU:n viralinen verkkosivusto. <https://ec.europa.eu/commission/presscorner/detail/en/qanda_21_1683>1.8.2024.Luettu 20.4.2025.

31 Gooch Ted. Strategies for Governing Data Quality, Accuracy, and Consistency. Verkkoaineisto. DZone.<https://dzone.com/articles/strategies-for-governing-data-quality-accuracy-and> 22.9.2022. Luettu 22.4.2025.

32 Thakur Sanjana. Top Trends in AI-Based Application Testing You Need To Know. Verkkoaineisto. DZone. <https://dzone.com/articles/10-top-trends-in-ai-based-application-testing-you> 31.8.2023. Luettu 22.4.2025.

33 Rachlin Ran. Essential Best Practices for Testing AI Applications. Verkkoaineisto.<https://ubertesters.com/blog/essential-best-practices-for-testing-ai-applications/> 22.7.2024. Luettu 22.4.2025.

34 AI Security: Risks, Frameworks, and Best Practices. Verkkoaineisto. Perception Point. <https://perception-point.io/guides/ai-security/ai-security-risks-frameworks-and-best-practices/> Luettu 23.4.2025.

35 Jie M. Zhang; Mark Harman; Lei Ma & Yang Liu. Machine Learning Testing: Survey, Landscapes and Horizons. Verkkoaineisto. IEEE. <https://ieeexplore.ieee.org/document/9000651> 17.2.2020. Luettu 23.4.2025.