Heini Ahven

TEKOÄLYN HYÖDYNTÄMINEN OHJELMISTOJEN LAADUNVARMISTUKSESSA

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2022

TIIVISTELMÄ

Ahven, Heini
Tekoälyn hyödyntäminen ohjelmistojen laadunvarmistuksessa
Jyväskylä: Jyväskylän yliopisto, 2022, 77 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Abrahamsson, Pekka

Tämän pro-gradu tutkielman aiheena on tekoälyn hyödyntäminen ohjelmistojen laadunvarmistuksessa. Tekoäly on jalkautunut teknologioihin, jota käytämme päivittäin ja sen hyödyntämismahdollisuudet laajenevat jatkuvasti. Viime vuosina tekoälyn hyödyntäminen ohjelmistokehityksessä ja ohjelmistojen laadunvarmistuksessa on ollut paljon käsitelty aihe informaatioteknologia-alan ajankohtaisjulkaisuissa. Ohjelmistokehityksessä asiakkaille pyritään kehittämään yhä nopeammin ja tehokkaammin ohjelmistotuotteita. Ohjelmistojen laadunvarmistus on kriittinen osa ohjelmistokehitystä. Laadunvarmistuksen avulla todennetaan, että ohjelmisto täyttää sille asetetut vaatimukset. Ohjelmistojen laadunvarmistuksessa on edelleen toimintoja, jotka ovat ihmisen manuaalisen työn tuotosta. Ohjelmistotestaus on tärkein ohjelmiston laadunvarmistuksen menetelmä. Testiautomaation avulla ohjelmistotestausta on saatu automatisoitua ja testit ovat osa lähes jokaisen ohjelmiston automaattista julkaisuprosessia. Testiautomaatio vaatii vielä vahvasti ihmisen ylläpitoa eikä se kykene mukautumaan itsenäisesti ohjelmiston muutoksiin. Tekoäly ja koneoppiminen voivat tuoda ohjelmistojen laadunvarmistusprosessiin älykkyyttä, jolloin ohjelmistotestauksen ihmisten tekemiä, manuaalisia toimintoja voidaan vähentää. Tutkimuksen avulla haluttiin selvittää, millä tavalla tekoälyä hyödynnetään ohjelmistojen laadunvarmistuksessa IT-organisaatioissa Suomessa. Tutkimus toteutettiin laadullisella teemahaastattelulla organisaatioille, joilla on jo kokemuksia tekoälystä ohjelmistojen laadunvarmistuksessa. Tutkimuksen tuloksissa kerrotaan, mitkä ovat ohjelmistojen laadunvarmistuksen osa-alueita, joihin tekoälyn avulla haetaan ratkaisuja sekä miten tekoälyä hyödynnetään näihin osa-alueisiin. Tulokset osoittavat myös ne haasteet, jotka ovat hidasteena tekoälyn jalkautumiselle osaksi ohjelmistotestausta.

Asiasanat: tekoäly, ohjelmistojen laadunvarmistus, ohjelmistotestaus, tekoäly ohjelmistojen laadunvarmistuksessa

ABSTRACT

Ahven, Heini
Utilization of artificial intelligence in software quality assurance
Jyväskylä: University of Jyväskylä, 2022, 77 pp.
Information Systems, Master's Thesis
Supervisor: Abrahamsson, Pekka

This master's thesis topic is how artificial intelligence can be utilized in software quality assurance. Artificial intelligence has spread widely to technologies, which we are using our daily basis and the opportunities for its utilization possibilities will expand continuously. During last years the utilization of artificial intelligence in software development and software quality assurance has been a popular topic in current publications of the Information Technology industry. In software development, the aim is to develop software products for customers faster and more efficiently. Software quality assurance is a critical part of a software development process. Software quality assurance is used to verify that software is fulfilling its requirements set to the software product. Quality assurance still includes functionalities that are carried out manually by human. Software testing is the most important method of software quality assurance. Test automation has been played a big role of automating manual testing, and tests are a part of software's continuous deployment. Test automation still needs human involving in its maintenance, and it's uncapable to conform independently to software changes. Artificial intelligence and machine learning may bring more intelligence to software testing, so human involving manual tasks can be decreased. The purpose of this study is to find out, how artificial intelligence is utilized in software quality assurance in IT-organizations in Finland. The study was conducted using qualitative interviews to organizations, which has some experiences of utilizing artificial intelligence in software quality assurance. The results of this study indicates which are the areas of software quality assurance for which solutions are sought with the help of artificial intelligence, and how artificial intelligence is utilized in these areas. The results also show the challenges that slow down the adoption of artificial intelligence as part of software testing.

Keywords: Artificial Intelligence, Software Quality Assurance, Artificial Intelligence in Software Quality Assurance, Software Testing

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ohjelmistoyritysten kilpailuedun saavuttamiseksi vaaditaan nykyään yhä nopeampaa arvon tuottamista asiakkaalle halvempaan hintaan, jolloin perinteiset ohjelmistojen testauskäytännöt eivät enää välttämättä mukaudu liiketoiminnan vaatimuksiin (Volk, 2021). Ohjelmiston laadunvarmistus- ja testaus ovat vahvasti ihmisen manuaalisen työn tuotosta. Testaaja havainnoi sovelluksen tilaa, toimii älykkäästi sekä tarkkailee ohjelmistoa vikojen löytämiseksi. Jotta laadunhallintaa voidaan parantaa ja kustannuksia alentaa, tulisi ohjelmiston testausta kehittää ottamalla käyttöön inhimillistä älykkyyttä jäljitteleviä ratkaisuja. (Santiago, King & Clarke, 2018.)

Tekoäly ohjelmistokehityksessä on nostettu ajankohtaisimpien IT-trendien joukkoon vuonna 2021 tutkimusyhtiö Gartnerin (2020) toimesta. Tekoäly on mediassa ja julkisissa keskusteluissa ollut aihe, jota on nykyisin lähes mahdoton välttää. Kaikki ovat kuulleet tekoälyn mainittavan kymmenen vuoden sisällä ainakin jossain asiayhteydessä tai vähintäänkin törmänneet siihen käyttäessään nykyteknologiaa. Tilastokeskuksen (2021) tekemän kyselytutkimuksen mukaan 16 prosenttia suomalaisista yrityksistä hyödyntää tekoälyä. Eniten tekoälyä hyödynnetään markkinoinnissa ja myynnissä (31 %). Lisäksi tuotantoprosessissa (22 %), tietoturvallisuudessa (18 %), yrityshallinnon prosesseissa (17 %) sekä yrityksen johtamisessa (17 %) hyödynnettiin tekoälyä. (Suomen virallinen tilasto, 2021.)

Tässä Pro gradu -tutkimuksessa on tavoitteena tutkia tekoälyn hyödyntämistä ohjelmistojen testauksessa ja selvittää, millä tavalla tekoälyä ja koneoppimista sovelletaan ohjelmistojen laadunvarmistuksessa IT-organisaatioissa Suomessa. Tekoäly ohjelmistojen laadunvarmistuksessa on ajankohtainen aihe IT-alan julkaisuissa ja aiheesta on tehty viimeisen viiden vuoden aikana useita tutkimuksia. Nykyinen tekoälyn hyödyntämien kohdistuu ohjelmistojen laadunvarmistusmenetelmistä pääosin ohjelmistotestaukseen, jonka vuoksi tässä tutkimuksessa ohjelmiston laadunvarmistusta käsitellään ohjelmistotestauksena. Foggin (2021) mukaan monet tekoälyyn perustuvat testausmenetelmät ovat kuitenkin vielä alkutekijöissään, eivätkä ne ole vielä levinneet laajalti yritysten

käyttöön. Tämän tutkimuksen avulla halutaan selvittää, missä tilassa tekoälyavusteinen testaus käytännön tasolla on.

Pro-gradun tutkimuskysymykset ovat:

- Miten tekoälyä hyödynnetään ohjelmistotestauksessa IT-organisaatioissa Suomessa?
- Millä tavalla tekoäly on muuttanut testausta organisaatiossa?

Tutkimuksen toteutustapa on laadullinen monitapaustutkimus ja empiirisessä osiossa tutkimusmenetelmänä käytetään teemahaastattelua. Haastattelut toteutetaan valittujen organisaatioiden testausasiantuntijoille. Tutkielman teoriaosiossa toteutetaan kirjallisuuskatsaus ohjelmistojen laadunvarmistuksesta, testauksesta sekä tekoälyn teoriasta. Ohjelmiston laadunvarmistuksen painotus tässä tutkimuksessa on ohjelmistotestaus. Kirjallisuuskatsauksen teoria on tukena empiiristä osiota toteutettaessa.

Tutkielman johdannon jälkeen kappaleissa 2–4 käsitellään aiempaa tutkimustietoa aihepiirin teemoista. Näissä osioissa perehdytään tekoälyyn, ohjelmiston laadunvarmistukseen ja erityisesti ohjelmistotestaukseen sekä tekoälyn hyödyntämismenetelmiin ohjelmistotestauksessa. Tutkimuksen empiirinen osio alkaa kappaleesta viisi, jossa kuvataan tutkimuksen toteutus ja käytetyt menetelmät. Kuudennessa kappaleessa kootaan teemahaastatteluista kertyneen tiedon pohjalta tutkimustulokset. Kappaleissa seitsemän ja kahdeksan käsitellään johtopäätökset, jatkotutkimusaiheet ja tutkielman yhteenveto.

Kirjallisuuskatsaukseen on haettu materiaalia käyttäen Google Scholaria, Googlen hakukonetta sekä Jyväskylän yliopiston tietokantaa. Hakuprosessissa on hyödynnetty lisäksi kansainvälisiä, tieteellisiä lehtijulkaisuja kuten IEEE Software ja Springer. Lähteet ovat yhdistelmä alan tieteellisiä konferenssijulkaisuja, tutkimusartikkeleita, kirjoja ja myös erilaisia internet-sivustoja, kuten blogikirjoituksia ja videoita. Lähteet arvioitiin niiden tuoreuden ja aiheen sopivuuden mukaan. Joitakin lähdemateriaaleja on myös löydetty ja hyödynnetty aiheeltaan sopiviksi katsottujen tutkimusten lähdeluettelosta. Aiheen ajankohtaisuuden ja tuoreuden vuoksi erilaisia internetsivustoja kuten organisaatioiden blogisivustoja hyödynnettiin, jotta tietoa konkreettisista tekoälyn hyödyntämisteknologioista saatiin. Aineiston etsimisessä käytettiin seuraavia hakusanoja: Artificial Intelligence, Software Quality Assurance, Artificial Intelligence in Software Quality Assurance, Software Testing.

Tutkimuksen teoriaosiossa hyödynnettiin harmaata kirjallisuuskatsausta johtuen tutkimusaiheen ajankohtaisuudesta. Tutkimusaiheesta ei ole vielä tehty montaa tieteellistä artikkelia, jolloin harmaasta kirjallisuudesta pystytään hyödyntämään tietoa ja teoriaa empiiristä osuutta varten. Garousin, Feldererin ja Mäntylän (2019) mukaan moniääninen kirjallisuuskatsaus eli Multivocal Literature Review (MLR) on systemaattisen kirjallisuuskatsauksen muoto, johon sisällytetään tieteellisten artikkeleiden lisäksi harmaata kirjallisuutta. Harmaa kirjallisuus tarkoittaa aineistoa, joka ei ole virallista, vertaisarvioinnin läpikäynyttä tieteellistä tietoa. Harmaata kirjallisuutta ovat muun muassa erilaiset

asiantuntijoiden ja kokemuspohjaiset blogipostaukset ja videot. Ohjelmistoala on hyvin käytännönläheinen ja alati muuttuva ala, jolloin ajankohtaisinta tietoa on usein saatavilla akateemisten foorumien ulkopuolella, ohjelmistoammattilaisten tuottamana. (Garousi, Felderer & Mäntylä, 2019.) Tästä syystä harmaa kirjallisuuskatsaus teoriaosuudessa on perusteltua.

Moniääninen kirjallisuuskatsaus eroaa systemaattisesta kirjallisuuskatsauksesta siten, että systemaattisessa kirjallisuuskatsauksessa hyödynnetään ainoastaan akateemisia, vertaisarvioituja artikkeleita, eli niin sanottua valkoista kirjallisuutta, johon sisältyvät myös tieteelliset konferenssit ja kirjat. Harmaa kirjallisuus määritellään olevan kaikkien hallinnon tasojen, tiedemaailman, liikeelämän ja teollisuuden tuottamaa painettua ja sähköistä materiaalia, niiden olematta kuitenkaan kaupallisten kustantajien omistuksessa. Tämä tarkoittaa sitä, että julkaiseminen ei ole edellä mainittujen tahojen ensisijaista toimintaa. (Garousi ym., 2019.) Tutkimuksen aiheen ollessa tuore ja aikaisemman tutkimustiedon ja konkreettisen kokemustiedon ollessa vajavaista, tutkimuksen luonne on ilmiötä kartoittava ja analysoiva.

Tutkielman tuloksena saatiin tietoa, millä tavoilla IT-organisaatiot ovat tekoälyä hyödyntäneet ohjelmistotestauksessa ja missä ohjelmistotestauksen vaiheissa tekoälyä pystytään hyödyntämään. Lisäksi saatiin tietoa siitä, mitkä asiat ovat vielä esteenä sille, että tekoälyavusteinen ohjelmistotestaus voisi jalkautua laajemmin osaksi organisaatioiden ohjelmistotestausta.

2 TEKOÄLY

Tässä luvussa käydään läpi tekoälyn määritelmää, sen alalajeja sekä lyhyesti tekoälyn kehitystä. Omana alalukunaan käsitellään tekoälyn yhtä käytetyintä alalajia, koneoppimista. Koneoppiminen on tämän päivän tekoälyn rakentamisessa yleisin tapa toteuttaa ratkaisuja (Tieturi Oy, 2021). Erityisesti tutkimuksen kannalta olennaiset teknologiat hyödyntävät koneoppimista, minkä vuoksi tähän alalajiin keskitytään erityisesti.

2.1 Tekoälyn määritelmä

Tekoäly eli AI (*eng. Artificial Intelligence*) on tietojenkäsittelytieteen ala, joka pyrkii tekemään tietokoneohjelmia, jotka jäljittelevät ihmisen älykkyyttä (Pati, 2021). Kayidin (2020) mukaan tekoälytieteen tavoitteena on tuottaa itseohjautuvia koneita ja järjestelmiä, jotka ovat vuorovaikutuksessa ympäristönsä kanssa, oivaltavat optimaalisen käyttäytymismallin ja kehittyvät ihmisen tavoin ajan myötä yrityksen ja erehdyksen kautta (Kayid, 2020). Shin (2011, s. 9–10) määritelmän mukaan tekoälytutkimuksessa kehitetään inhimillisen käyttäytymisen laskennallisia malleja järjestelmiin, jotka havainnoivat, päättelevät, oppivat, assosioivat, tekevät päätöksiä ja ratkaisevat monimutkaisia ongelmia ihmisen tavoin. Tekoälyjärjestelmä ei vaadi esiohjelmointia, vaan se käyttää algoritmeja, jotka voivat toimia itsenäisesti ikään kuin oman älynsä kanssa ja parantavat suoritustaan jatkuvasti oppien. (Pati, 2021.) Tekoälylle on olemassa monenlaisia määritelmiä, eikä kirjallisuudessa tuoda yhtä ainoaa määritelmää esiin. Yhteistä teorioissa kuitenkin on, että tekoäly tarkoittaa tietokoneohjelmaa tai koneita, jotka pystyvät suorittamaan ihmiselle ominaisia, älykkäinä pidettäviä toimintoja.

Tekoäly ei ole kuitenkaan sama kuin ihmisen älykkyys. Koneälykkyys ei ole tiedostettua prosessia, vaan on puhtaasti mekaanista, kun taas ihmisen äly on fyysistä ja psykologista. Tekoälyssä ei ole myöskään sosiaalisuutta, tunneälyä tai luovuutta, joita ihmisellä on. Ihmisen ajatusvirta ja älykkyys on dynaamista, liikkuvaa tietoisuutta, kun taas koneälykkyys ovat tietokoneen toimintoja. (Ning &

Yan, 2010.) Tekoälyn odotetaan koskettavan jollakin tavalla lähes jokaista alaa lähitulevaisuudessa. Tällä hetkellä tekoälyä on hyödynnetty esimerkiksi lääketieteellisissä diagnooseissa, kaupankäyntialustoissa, robottiohjauksessa ja erilaisissa etäkartoituksissa. Sitä on käytetty monien toimialojen kehittämiseen ja edistämiseen kuten rahoitusalaan, terveydenhuoltoon, koulutukseen, kuljetukseen ja robotiikkaan. (Kayid, 2020.) Voidaan siis perusteellisesti todeta, että tekoäly tulee ulottumaan kaikkialle ja se on jo nyt vahvasti läsnä ihmisten arjessa.

Läpi tekoälyn historian tutkijat ovat yrittäneet kehittää erilaisia tapoja määritellä tekoälyä (Russell, Chang, Devlin, Dragan, Forsyth, Goodfellow &. Wooldridge, 2022, s. 19). Shin (2011, s. 10) mukaan tutkijat kategorisoivat tekoälytieteen kahteen kategoriaan: symboliseen älykkyyteen ja laskennalliseen älykkyyteen. Symbolinen älykkyys, jota pidetään perinteisenä tekoälynä, ratkaisee ongelmia tietoon perustuvan päättelyn avulla, kun taas laskennallinen älykkyys tekee saman perustuen esimerkkidatasta opetettujen yhteyksien avulla. (Shi, 2011, s. 10.) Tekoälyssä on paljon erilaisia alalajeja ja luokitteluja. Tutkijat luokittelevat tekoälyn pääalueet eri tavoin riippuen tekoälyyn liitetystä kontekstista, mikä tekee tekoälylajien luokittelun haastavaksi.

Aineiston perusteella voidaan todeta, että yksiselitteistä määritelmää tai kategorisointia tekoälystä ei ole kehitetty. Myöskään alan tutkijoiden keskuudessa ei ole yhtä hyväksyttyä määritelmää tekoälylle, koska tekoäly kehittyy nopeasti ja se täytyy näin ollen määrittää jatkuvasti uudelleen (Elements of AI, 2021). Vuosina 2010–2019 tekoälytutkimusten kolme suosituinta alalajia olivat koneoppiminen, konenäkö sekä luonnollisen kielen käsittely tässä järjestyksessä (Russell ym, 2022, s. 45).

2.2 Tekoälyn kehitys

Tekoälytieteen katsotaan saaneen alkunsa matemaatikko Alan Turingin 1950-luvulla kehittämässään Turing testissä, jonka avainkysymyksenä on ”voiko kone ajatella?”. Testin tarkoitus on selvittää, erottaako ihminen olevansa vuorovaikutuksessa ihmisen vai koneen kanssa perustuen koneen kanssa käytyyn vuorovaikutukseen. Mikäli vuorovaikuttaja ei kykene erottamaan koneen vastausten perusteella, onko hän vuorovaikutuksessa ihmisen vai koneen kanssa, kone on älykäs. (Russell ym., 2022, s. 20; Elements of AI, 2021.) Turing ikään kuin keksi tekoälyn, mutta ensimmäinen merkittävä tekoälyä käsittelevä tapahtuma on katsottu olevan 1956-luvulla järjestetty tutkimusprojekti Darthmouthin yliopistossa (Bruderer, 2016; Brunette, Flemmer & Flemmer, 2009).

Yliopiston apulaisprofessori John McCarthy järjesti kolmen muun vanhemman tutkijan kanssa konferenssin, jonka rahoitusehdotuksessa todettiin, että tutkimuksen tavoitteena on edetä olettamuksen pohjalta, jossa jokainen oppimisen tai älykkyyden näkökulma voidaan kuvata niin tarkasti, että kone voi simuloida sitä. Tämä saavutetaan selvittämällä, miten saada koneet käyttämään luonnollista kieltä, muodostamaan abstraktioita ja konsepteja, ratkaisemaan ongelmia ja kehittämään itseään. (Kaplan, 2016, s. 13–16.)

Seuraavina vuosina kehitettiin useita muita tekoälyyn viittaavia tietokoneohjelmia ja metodologeja, kuten universaali ongelmanratkaisukone General Problem Solver vuonna 1959, luonnollisen kielen käsittelyohjelma Eliza vuonna 1966 sekä shakkitietokone Deep Blue vuonna 1997. 1980-luvulla tutkijat alkoivat ymmärtää, että tekoälyn kehittäminen on haastavampaa kuin aluksi luultiin. Tämän huomioiden, robotiikka-asiantuntija Rodney Brooksin mielestä tutkijoiden tulisi kehittää itsenäisiä moduuleja, jotka perustuvat ihmisaivojen eri osa-alueisiin. Näitä ovat esimerkiksi suunnittelu- ja muistimoduuli, jotka yhdessä muodostavat älykkyyden. Hän uskoi, että tällä tavalla tekoälyn kehittämisessä edistyttäisiin. (Brunette ym., 2009.)

1990-luvulla tekoälytutkimus alkoi siirtyä enemmän verkkopohjaiseen ympäristöön johtuen internetin kehityksestä. Viimeistään 2000-luvulla tietotekniikan räjähdysmäisen kehityksen seurauksena tekoäly on jalkautunut ohjus- ja varoitusjärjestelmiin sekä nykyaikaisiin aseisiin. Tekoäly on myös ihmisten kodeissa älylaitteissa ja kodinkoneissa, jotka hyödyntävät puheen- ja tekstintunnistusta. Tekoäly on tällä hetkellä läsnä kaikilla aloilla ja se on muuttanut ja tulee muuttamaan jatkossa ihmisten elämää. (Ning & Yan, 2010.)

2.3 Tekoälyn lajeja

Russell ym. (2022) perustelevat Turingin testiin pohjaten, että kuusi tieteenalaa muodostaisivat suurimman osan tekoälystä. Nämä ovat luonnollisen kielen käsittely, tiedon esittäminen, asiantuntijajärjestelmä, koneoppiminen, konenäkö ja puheentunnistus sekä robotiikka. (Russell ym., 2022, s. 20.) Näiden ominaisuuksien avulla älykäs kone pystyy kommunikoimaan, vastaanottamaan tietoa ja toimimaan myös fyysisesti. Kaplan (2016, s. 50–60), Ning ja Yan (2010) sekä Kayid (2020) mainitsevat myös robotiikan, konenäön, puheentunnistuksen sekä luonnollisen kielen prosessoinnin olevan tekoälytutkimuksen pääalueita. Asiantuntijajärjestelmät, neuroverkot ja syväoppiminen mainitaan myös monessa tutkimuksessa osana tekoälyn pääalueita.

Tässä osiossa käsitellään luonnollisen kielen käsittelyä, konenäköä, puheentunnistusta ja robotiikkaa. Koneoppimista käsitellään perusteellisemmin omassa osiossaan, koska suurin osa ohjelmistotestaukseen ja moneen muuhun tieteenalaan tarkoitetut teknologiat pohjautuvat siihen.

2.3.1 Luonnollisen kielen käsittely

Luonnollisen kielen käsittely, eli NLP (*eng. Natural Language Processing*) on kone tai järjestelmä, joka käsittelee ihmisen tuottamaa, tekstimuotoista luonnollista kieltä (Kaplan, 2016, s. 63). Luonnollisen kielen käsittelytoiminto luo tekstiä, vastaa käyttäjän kysymyksiin, tunnistaa tekstiä kontekstistaan sekä luokittelee ja kääntää tekstiä (Kayid, 2020). Luonnollisen kielen käsittelyn tarkoitus on pystyä kommunikoimaan ihmisen kanssa ja oppia, mitä ihmiset kirjoittavat (Russell ym., 2022, s. 874).

Voidaan sanoa, että iso osa tekoälyjärjestelmien algoritmeista on luonnollisen kielen prosessointia, koska tekoälyn halutaan suorittavan toimintoja, joita ihmiset kykenevät määrittelemään luonnollisella kielellä. Ihmisen kommunikointitapa on kieli, puhuttu ja kirjoitettu, jolloin älykkään järjestelmän tulee osata ymmärtää kieltä ja reagoida siihen. Verkkokauppojen chatbotit ovat yksi konkreettinen esimerkki teknologiasta, joka hyödyntää luonnollisen kielen käsittelyä.

2.3.2 Konenäkö

Konenäkö (*eng. Visual AI*) pyrkii jäljittelemään ihmisnäköä tai laajentamaan sitä. Sen tarkoitus on nähdä ja tulkita visuaalisia elementtejä. Konenäkö koostuu valonlähteestä, kohteesta, kamerasta, tietokoneesta ja siinä toimivasta kuvankäsittelyohjelmasta, joka tulkitsee kuvaa. Konenäköä käytetään tunnistamaan esineitä, paikkoja, ihmisiä, kirjoitusta ja toimintaa kuvissa. (Neittaanmäki & Tuominen, 2019; Kaplan, 2016, s. 54, 56.)

Konenäön avulla tekoälyjärjestelmä voidaan kouluttaa tunnistamaan kuvien avulla haluttuja asioita. Monien nykyaikaisten älypuhelimien käyttäjiltä löytyy puhelimestaan Google Lens. Se on kuvantunnistussovellus, joka hyödyntää puhelimen kameraa tunnistaessaan kameralla osoitettuja asioita. Käyttäjä voi kuvata puhelimen kameralla mitä tahansa asiaa, kuten tekstiä, tuotteita tai luontoa. Google Lens analysoi kuvan ja hakee vastaavuuksia verkosta. Sen avulla voi selvittää, mikä lintulaji istuu edessä parasta aikaa, mistä löytää samanlaiset kengät, kuin ohikulkijalla tai kääntää vieraskielisen ohjeen omalle kielelle.

2.3.3 Robotiikka

Robotiikka ei suoranaisesti ole tekoälyä, mutta robotti on fyysinen toimija, joka käyttää tekoälyä suorittaakseen toimintoja. Robotit voivat olla yksinkertaisia tehtäviä suorittavia laitteita tai monimutkaisia järjestelmiä. Ne voivat tunnistaa ympäristöään, tehdä päätöksiä ja mukauttaa suunnitelmiaan havaintojensa mukaisesti. (Kaplan, 2016, s. 49–51.)

Robotit on varustettu sensoreilla ja antureilla, joiden avulla ne voivat havainnoida ympäristöään ja reagoida ympäristön muutoksiin. Sensorit ovat kameroita, tutkia, lasereita ja mikrofoneja. (Russell ym., 2022, s. 932–933.) Esimerkkejä nykyajan autonomisista roboteista ovat itseajavat ajoneuvot, droonit ja robotti-imurit.

2.3.4 Puheentunnistus

Puheentunnistus on joukko kieli- ja puheteknologiaan kuuluvia tunnistusmenetelmiä, joiden avulla tietokone kykenee poimimaan, tunnistamaan ja analysoimaan ihmisen puhetta (Neittaanmäki & Tuominen, 2019). Puheentunnistus voi muuttaa puheen tekstimuotoon ja vastaavasti muuntamaan tekstin puheeksi (Kayid, 2020). Puheentunnistus luokitellaan usein luonnollisen kielen

prosessoinnin alle. Kuten Russell ym. (2022) mainitsee, luonnollisen kielen prosessoinnin yhtenä tarkoituksena on koneiden kommunikointi ihmisen kanssa. Monissa tilanteissa ihminen käyttää puhetta ollakseen vuorovaikutuksessa tietokoneen kanssa. Tunnetut tekoälyjärjestelmät Google Voice ja Applen Siri ovat esimerkkejä puheentunnistusta hyödyntävistä järjestelmistä.

2.4 Koneoppiminen

Koneoppiminen (*eng. Machine Learning*) on tekoälyn alalaji, jossa kone tai järjestelmä oppii ja parantaa toimintaansa käsittelemänsä datan avulla (Jordan & Mitchell, 2015). Neittaanmäen ja Tuomisen (2019) mukaan koneoppimisen tarkoituksena on saada ohjelmisto toimimaan jatkuvasti paremmin pohjatiedon ja käyttäjän toiminnan perusteella. Koneoppimisessa ohjelmisto tai kone oppii toistuvista tapahtumista ilman ihmisen osallistumista. (Neittaanmäki & Tuominen, 2019.)

Tekoäly ja koneoppiminen saatetaan sekoittaa helposti keskenään, sillä molemmissa kone tai ohjelmisto oppii itsenäisesti oppimisdataan perustuen ja tulee älykkäämmäksi. Koneoppiminen on yksi tekoälyn alalajeista, mutta tekoäly itsessään on osa tietojenkäsittelytieteen alaa (Elements of AI, 2021). Tekoälyn ja data-analytiikan johtava ajankohtaislehti Analytics Insight määrittelee koneoppimisen ja tekoälyn eron seuraavasti: tekoälyssä tehdään älykkäitä järjestelmiä suorittamaan mitä tahansa tehtävää.

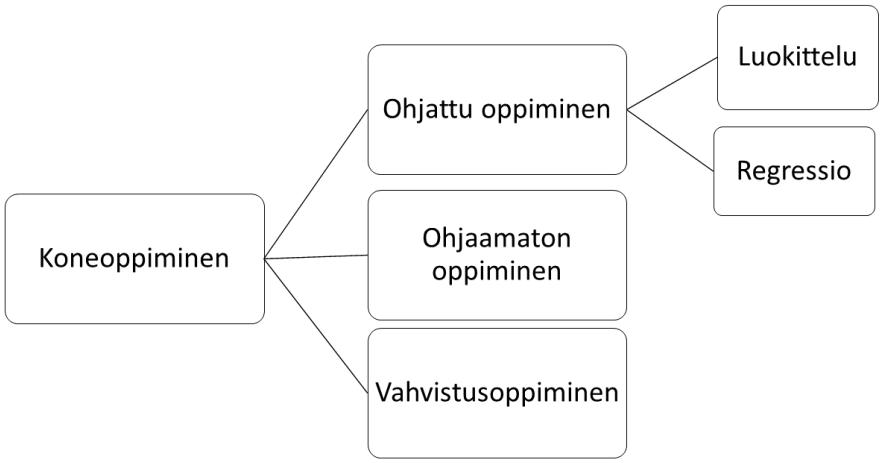
Koneoppimisessa opetetaan dataa sisältävät järjestelmät ja koneet suorittamaan tiettyä toimintoa ja antamaan tästä toiminnosta tarkka tulos. Tekoälyn tarkoitus on simuloida ihmisen älykkyyttä kokonaisvaltaisesti. Koneoppimisessa ohjelmisto pystyy käsittelemään suuria määriä oppimisdataa, mutta vain tiettyyn tarkoitukseen, esimerkiksi tunnistamaan kuvia erilaisista koirista. Applen Siri, asiakaspalvelussa käytettävät chatbotit ja online-pelit ovat esimerkkejä tekoälysovelluksista, kun taas suosittelujärjestelmät, Googlen hakualgoritmit ja Facebookin kaveriehdotukset ovat koneoppimiseen perustuvia toimintoja. (Pati, 2021.)

Ensimmäisten koneoppimisen ilmentymien katsotaan ilmaantuneen vuonna 1946, jolloin Chicagon yliopiston tutkijat havaitsivat, että aivohermosolujen verkosto pystytään mallintamaan loogisten ilmaisujen avulla. Vaikka aivot itsessään ovat pehmeää massaa, niiden tietoliikenne voidaan katsoa olevan digitaalista. (Kaplan, 2016. s. 32–35) Oppiminen tapahtuu kokemuksen ja harjoittelun seurauksena, tiedon ja kokemuksen kerryttämisellä, suorituskyvyn parantamisella, sääntöjen selvittämisellä sekä sopeutumalla oppimisympäristöön. (Kaplan 2016, s. 27; Shi, 2011, s. 18.)

2.4.1 Koneoppimismallit

Oppiminen on monisyisempi tapahtuma, kuin pelkästään datan tallentuminen tietokantaan. Oppimisen tulos tulee pystyä havainnollistamaan ja

hyödyntämään jollain tavalla. Tietokoneohjelmat oppivat käyttäytymismallit mitä erilaisimpien datamuotojen avulla kuten videoista, raporteista, Facebook-tykkäyksistä, mainosten klikkauksista tai luottokorttitapahtumista. (Kaplan 2016, s. 27.) Suosituimmat koneoppimisalgoritmit ovat ohjaamaton- ja ohjattu-, sekä vahvistusoppiminen. Kuviossa 1 havainnollistetaan koneoppimismalleja mukaillen Houranin ym. (2019) kuviota.



KUVIO 1 Koneoppimisalgoritmit (Hourani ym., 2019.)

Ohjatussa oppimisessa käytetään ihmisen merkitsemää oppimisdataa (Santiago ym., 2018). Tarkoituksena on opettaa kone tekemään jaotteluja keskenään samankaltaisille aineistoille. Ohjattu oppiminen jaetaan kahteen luokkaan, luokitteluun ja regressioon. Luokittelussa data voidaan jakaa erillisiin ryhmiin. Regressiossa data on jatkuvaa. (Neittaanmäki & Tuominen, 2019.) Esimerkiksi sähköpostin roskapostilajittelu, kasvojentunnistus kuvien päällä sekä lääketieteelliset diagnosointijärjestelmät ovat luokittelua, kun taas jonkin tuotteen hinnan määrittely on regressiota. (Jordan & Mitchell, 2015.)

Ohjaamaton oppiminen jäljittelee ihmisen oppimista. Siinä jalostamaton tieto viedään oppimisalgoritmille, joka pyrkii muodostamaan syötteiden (*eng. Input*) välillä olevia malleja ja suhteita. Syötteet ryhmitellään ryhmiksi tai ryppäiksi (*eng. Clustering*) niiden samankaltaisuuksien perusteella. Tämä tarkoittaa, että syötteellä on enemmän samankaltaisia ominaisuuksia samaan ryhmään kuuluvien syötteiden kanssa, kun muiden ryhmien syötteiden kanssa. (Santiago ym., 2018; Neittaanmäki & Tuominen, 2019.) Esimerkiksi asiakkaiden segmentointi on ohjaamattoman oppimisen tuotosta. Ohjaamattomassa oppimisessa syötteille ei ole ennalta määrättyjä luokkia, eikä oppimisalgoritmissa ole malleille myöskään oikeita vastauksia.

Kolmas paljon käytetty koneoppimisen malli on vahvistusoppiminen. Jordanin ja Mitchellin (2015) mukaan se perustuu algoritmeihin, jotka pystyvät oppimaan käyttäen palkitsemisjärjestelmää. Koulutusesimerkkien sijaan, vahvistusoppimisessa oppimisdata antaa tiedon siitä, onko toiminto oikein vai väärin. Hyödylliset toiminnot palkitaan, kun taas vääristä toiminnoista voidaan rankaista jollakin tavalla. (Jordan & Mitchell, 2015.) Tällöin kone pyrkii tekemään

asioita, joista seuraa palkinto ja välttämään rankaisua aiheuttavia toimintoja. Itseajavat autot ovat esimerkki vahvistusoppimisesta. (Neittaanmäki & Tuominen, 2019.)

Koneoppimisessa oppimisdatan perusteella tapahtuva oppiminen ei välttämättä tuota sellaisia tuloksia, jota alun perin on haettu. Esimerkiksi rikosten uusimisen ennustemallit riippuvat iästä ja rikoshistoriasta, mutta eivät eksplisiittisesti ihmisen ihonväristä. Kuitenkin kun nämä seikat korreloivat myös ihonvärin kanssa, voi koneoppimismalli muodostaa säännön, jonka mukaan henkilö ennustetaan pidätetyksi, koska hän on tummaihoinen. Tämä malli voi jäljitellä oikein mallin ennusteita, mutta se ei ole välttämättä sitä, mitä alkuperäisen mallin tulisi laskea. (Rudin, 2019.) Tästä syystä oppimisen malli on kehittynyt tummaihoisista henkilöistä koostuvan oppimisdatan perusteella olettamaan tumman ihonvärin olevan syynä henkilön potentiaaliselle pidätykselle. Tällöin koneoppimisalgoritmin lopputulos ei palvele alkuperäistä tarkoitustaan ja on rajusti syrjivä.

Monet koneoppimismallit ovat jo lähestulkoon mustalaatikkomalleja, jolloin ihminen ei voi nähdä tai ymmärtää niiden toimintaan perustuvia algoritmeja. Tämä johtuu siitä, että oppimisdataa kertyy hyvin paljon. Mustalaatikkomalli on usein toiminto, joka on liian hankala ihmisen ymmärrettäväksi tai se on patentoitu. Syväoppimismallit ovat usein mustia laatikoita, koska ne ovat hyvin rekursiivisia. (Rudin, 2019.) Rudinin ja Radinin (2019) väittävät, että lähes jokainen monimutkainen datajoukko koneoppivassa mustalaatikkomallissa sisältää jonkinlaisia puutteita. Nämä voivat olla seurausta puuttuvista tietomääristä, tietoaineiston systemaattisista virheistä kuten virheellisestä tietojen koodaamisesta tai tiedonkeruuongelmista. (Rudin & Radin, 2019.)

Tämä on ongelmallista, sillä harjoitusdataan voi sisältyä sellaisia tapauksia, jotka voivat saada koneoppimisalgoritmin käyttäytymään väärällä tavalla. Vääränlainen toiminta täytyy pystyä tunnistamaan ja poistamaan. Mustalaatikkomallissa epätoivotun oppimistuloksen mekanismia ei välttämättä ymmärretä, jolloin sen toimintaan ei voida puuttua. Tästä syystä kriittisiin päätöksentekoon tarkoitettujen koneoppimismallien ei tulisi noudattaa mustalaatikkomallia, vaan tulkittavia ja läpinäkyviä oppimismalleja.

Tulkittavassa koneoppimisessa päätöksenteon taustalla olevat algoritmit ovat ihmisen tulkittavissa ja ymmärrettävissä (Murdoch, Singh & Kumbier, 2019). Gaon, Taon, Jien ja Lun (2019) mukaan tekoälytoimintojen laatu riippuu vahvasti valituista koulutusmalleista, syötetyn tiedon laadusta sekä koneoppimismallin koulutusprosessista ja menetelmistä. Jotta tekoälysovelluksen toiminta on läpinäkyvää ja sitä voidaan ymmärtää, tulisi näihin laatutekijöihin kiinnittää huomiota.

2.4.2 Neuroverkot ja syväoppiminen

Koneoppimisen olennainen alalaji on neuroverkot. Siinä tietokoneohjelma jäljittelee todellisten neuroverkkojen toimintaperiaatteita Ihmisaivot ovat todellisen maailman esimerkki hermoverkoista (Kaplan, 2016, s. 28.) Neuroverkot perustuvat yhdistävään laskentaan ja ne ovat informaation käsittelyn, matematiikan tai

laskennan malleja. Neuroverkkoja käytetään kuvantunnistuksessa, konenäössä, puheentunnistuksessa, kieltenkääntäjissä, peleissä ja lääketieteellisissä diagnooseissa. Neuroverkko koostuu syöte- ja ulostulokerroksesta, ja niiden välissä olevista piilokerroksista, jotka koostuvat neuroneista. (Neittaanmäki & Tuominen, 2019.)

Syväoppiminen tarkoittaa sitä, että neuroverkoissa käytetään monia piilokerroksia, joilla kullakin on oma tehtävänsä. Syvät neuroverkot ovat piirteenmuodostamiseen pystyviä monikerroksisia neuroverkkoja. Jordanin ja Mitchellin (2015) mukaan syväoppimisjärjestelmä voi sisältää miljardeja neuroneita ja näin ollen myös miljoonia muutettavia parametreja. Niitä voidaan kouluttaa Internetissä saatavilla olevien erittäin suurten kuva-, video- ja puhenäytteiden kokoelmien avulla.

Tällaisilla laajamittaisilla syväoppimisjärjestelmillä on ollut viime vuosina suuri vaikutus tietokonenäön ja puheentunnistuksen alalla, missä ne ovat parantaneet merkittävästi tekoälyn suorituskykyä. (Jordan & Mitchell, 2015.) Neittaanmäen ja Tuomisen (2019) mukaan syväoppimisessa on kuitenkin haasteena sen tarvitseman massiivisen datan määrä. Liian vähäinen datan määrä johtaa siihen, että verkot ylioppivat helposti, eivätkä niiden tulokset yleisty ennalta tuntemattomiin havaintoihin. (Neittaanmäki & Tuominen, 2019.)

3 OHJELMISTOJEN LAADUNVARMISTUS

Ohjelmistotestaus käsittää laajan joukon erilaisia ohjelmiston laadunvarmistusmenetelmiä, manuaalisia ja automatisoituja. Tekoälyn hyödyntäminen ohjelmistotestauksessa ja laadunvarmistuksessa on kehittynyt testiautomaation ja ohjelmistorobotiikan avulla. Koska asiat liittyvät vahvasti toisiinsa, on tärkeää ymmärtää testiautomaation, ohjelmistorobotiikan sekä tekoälyn eroavaisuudet ohjelmistotestauksessa. Tässä kappaleessa käsitellään ohjelmiston laatu sekä ohjelmistotestaus käsitteinä sekä testaustasot.

3.1 Ohjelmiston laatu

Ohjelmiston vaatimukset ovat kuvaus siitä, mitä palveluita ohjelmiston tulee tarjota. Vaatimukset pohjautuvat asiakkaiden tarpeisiin. Esimerkki asiakkaan vaatimuksesta ohjelmiston toimintaan voi olla tilauksen tekeminen tai tiedonhaku. Ohjelmiston vaatimukset jaetaan käyttäjävaatimuksiin sekä järjestelmävaatimuksiin. Käyttäjävaatimukset laaditaan luonnollisella kielellä kuvailemaan, mitä järjestelmän odotetaan tarjoavan sen käyttäjille ja minkä ehtojen mukaan sen tulee toimia. Järjestelmävaatimukset ovat yksityiskohtaisempia kuvauksia järjestelmän toiminnoista, palveluista ja sen toiminnallisista rajoitteista. Järjestelmävaatimusten dokumentoinnissa kuvataan tarkasti, mitä järjestelmäkehityksessä tulee toteuttaa. (Sommerville, 2016, s. 102.) Laadukkaan ohjelmiston tulee ensisijaisesti toimia asiakkaan asettamien vaatimusten mukaisesti.

Ohjelmiston kokonaisvaltainen laatu voidaan saavuttaa ymmärtämällä ensin, mitä laadulla tarkoitetaan ja mitä toimenpiteitä on tehtävä ohjelmiston laatustandardien saavuttamiseksi (Goericke, 2020). ISO/IEC/IEEE 24765:2017 -standardi määrittelee termin ohjelmiston laatu (*eng. Software Quality*) tarkoittavan ohjelmistotuotteen kykyä täyttää sille asetetut ja oletetut tavoitteet sille tarkoitetussa toimintaympäristössä. Standardi määrittelee myös ohjelmiston laadunvarmistus (*eng. Software Quality Assurance, SQA*) tarkoittavan joukkoa käytänteitä, joissa arvioidaan ohjelmistotuotteiden kehittämiseen ja muokkaamiseen

tarkoitettujen ohjelmistoprosessien noudattamista ja asianmukaisuutta. Termi käsittää lisäksi laadunvarmistuksen avulla tavoiteltavien tulosten tason. (ISO, 2017.)

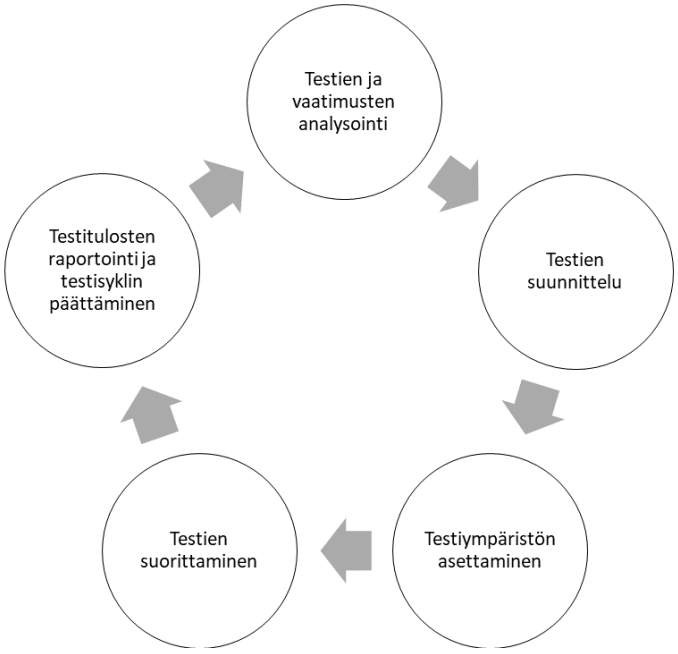
Laadunvarmistus käsittää kaikki ne keinot, joiden avulla lisätään luottamusta kehitettyyn tuotteeseen sekä tekniikat tuotteiden analysointiin (Wagner, 2013). Laadukas ohjelmisto sisältää monta eri osa-aluetta. Wagner (2013) luettelee laadultaan hyvän ohjelmiston täyttävän seuraavat kriteerit: toiminnallinen soveltuvuus, luotettavuus, suorituskyvyn tehokkuus, käytettävyys, turvallisuus, ylläpidettävyys, siirrettävyys sekä yhteensopivuus. Nämä kriteerit jaotellaan ohjelmiston toiminnallisiksi ja ei-toiminnallisiksi ominaisuuksiksi.

Ohjelmiston toiminnalliset vaatimukset kuvaavat sitä, mitä ohjelmiston tulee käyttäjälle tarjota, miten se reagoi sille annettuihin syötteisiin ja kuinka ohjelmiston tulisi toimia tietyissä tilanteissa. Kuten nimikin kertoo, toiminnallisuus viittaa siihen, mitä järjestelmän pitäisi tehdä. (Sommerville, 2016, s. 105.) Toiminnalliset vaatimukset ovat yleensä ohjelmiston näkyviä osia ja ne määritellään usein asiakkaan toimesta käyttäjätarinoiden (*eng. User stories*) avulla. Esimerkkinä toiminnallisesta vaatimuksesta voi olla, että käyttäjän tulee pystyä etsimään kaikkien osallistujien nimet tapahtumasivustolta.

Ei-toiminnalliset vaatimukset ovat usein käyttäjältä piilossa olevia ominaisuuksia ja ne koskevat yleensä järjestelmää kokonaisuutena. Nämä ominaisuudet ovat ohjelmiston toiminnan kannalta kriittisiä. Erimerkiksi pankkijärjestelmä ei voi toimia, ellei se täytä sille asetettuja turvallisuusvaatimuksia. (Sommerville, 2016, s. 108.) Ei-toiminnallisia vaatimuksia ovat muun muassa turvallisuus, suorituskyky, käytettävyys, ylläpidettävyys ja siirrettävyys.

3.2 Ohjelmistotestaus

Ohjelmistotestaus on tärkein ohjelmiston laadunvarmistustekniikka, jonka tarkoituksena on löytää ohjelmiston virheet ennen ohjelmiston käyttöönottoa (Wagner, 2013; Sommerville, 2016, s. 227). Se on joukko käytänteitä ja prosesseja, joiden avulla tarkistetaan, vastaavatko ohjelmiston tulokset siltä odotettuja tuloksia ohjelmiston vaatimusten mukaisesti (Hourani et al., 2019). Testauksen avulla varmistetaan myös, että ohjelmisto täyttää sen tekniset- ja liiketoiminnalliset vaatimukset (Quadri & Sheikh, 2010). Testaus on vahvasti dynaaminen laadunvarmistustekniikka, jossa ohjelmisto suoritetaan keinotekoisella datalla ja testiajon tuloksista tarkastetaan virheet ja poikkeavuudet (Wagner, 2013; Sommerville, 2016, s. 227). Kuviossa 2 on esitetty ohjelmistotestauksen elinkaari mukaillen Hooda & Chhillar (2016) testausprosessin määritelmiä.



KUVIO 2 Ohjelmistotestauksen elinkaari (Hooda & Chhillar, 2016)

Testauksessa joudutaan tekemään usein montaa eri vaihetta samanaikaisesti tai palaamaan edelliseen vaiheeseen, jolloin testauksen elinkaari ei välttämättä etene vaihe kerrallaan peräkkäin. Projektien aikapaineiden vuoksi testit saatetaan suorittaa ennen kuin kaikki testitapaukset on suunniteltu. (Hambling, Morgan & Samaroo, 2010.) Elinkaaren mukaan ensimmäinen vaihe on testien ja ohjelmistovaatimusten analysointivaihe. Tämä vaihe sisältää ohjelmiston toiminnallisten ja ei-toiminnallisten toimintojen analysoinnin. Analysointivaiheessa pyritään tunnistamaan todelliset ja odotetut testaustulokset sekä ohjelmiston vaatimusten toteutuminen. Nämä ovat usein ei-toiminnallisia vaatimuksia. Suoritettavat testit pyritään tunnistamaan ja priorisoimaan. (Hooda & Chhillar, 2016.)

Testien suunnittelu kattaa koko ohjelmiston testausstrategian. Tavoitteena on suunnitella mahdollisimman pieni määrä testitapauksia kattamaan mahdollisimman laaja joukko testattavia ominaisuuksia (Hambling ym., 2010). Testisuunnitelman laajuuden määrittää McLeodia ja Everettiä (2007) mukaillen, mitä sovelluksia testataan ja miksi, mitä ohjelmiston tukikerroksia testataan sekä kuinka iso osa koko ohjelmistokehitystyön elinkaaresta testataan. Testisuunnitelman tulisi kattaa lisäksi testiympäristön tiedot, luettelo kaikista testitapauksista, testausaikataulun ja testitiimin roolit. (McLeod & Everett, 2007, s. 70.)

Testiympäristön asettamisvaiheessa konfiguroidaan ympäristö, jossa testit voidaan suorittaa ja varmistetaan sen toimivuus. Kun testit on koodattu, valmistetaan testiversio, jossa testaajan on aloitettava testin suorittaminen. Tässä vaiheessa tunnistetaan ja arvioidaan myös testiautomaatiotyökalut ja suunnitellaan testien komentosarja (Hooda & Chhillar, 2016; Rana, Goswami & Maheshwari, 2019). Usein automaatiotyökalut ovat osa versionhallinnan tai DevOps-työkalun putkeen lisätään tehtäviä, jotka suorittavat testit samalla, kun ohjelmistoa ollaan viemässä versionhallintaan tai muuhun julkaisualustaan. DevOps (*eng. Development and Operations*) on toimintamalli, joka pyrkii automatisoimaan

ohjelmistokehityksen julkaisuprosessin. Siinä ohjelmiston uudet versiot päivitetään jatkuvassa julkaisuputkessa, johon sisällytetään myös automatisoidut testit. (Leite, Rocha, Kon, Milojicic & Meirelles, 2019.)

Testien suorittamisvaiheessa testit suoritetaan ohjelmiston testitapausten mukaisesti. Mikäli odotetut testitulokset eivät täsmää todellisten tulosten kanssa, virheet avataan, tutkitaan ja korjataan. Viimeisessä vaiheessa testitulokset raportoidaan, ohjelmiston viimeisin versio julkaistaan testien onnistuttua ja testisykli päätetään. Testien sulkeminen sisältää kaikki tiedot, joiden avulla varmistetaan, että järjestelmä-, integraatio- ja käyttäjähyväksyntätestaukset ovat toteutuneet vaatimusten mukaisesti. Lisäksi tehdään päätös siitä, onko kaikki vaatimukset testattu ja onko löytynyt virheitä, jotka edellyttävät toimenpiteitä. (Hooda & Chhillar, 2016; Rana ym., 2019.)

Tässä vaiheessa saatetaan esimerkiksi arvioida, onko testikattavuus täyttynyt. Jos kriteerinä on ollut 80 % testikattavuus, mutta testiajon jälkeen testikattavuusprosentti onkin 75 %, tehdään lisää testejä tai alennetaan kattavuusprosenttia. Vaikka jokainen virhetapaus vaatii selvittelyn, korjaavia toimenpiteitä ei kaikissa tapauksissa ole tarpeen tehdä (Hambling ym., 2010).

3.3 Testaustasot

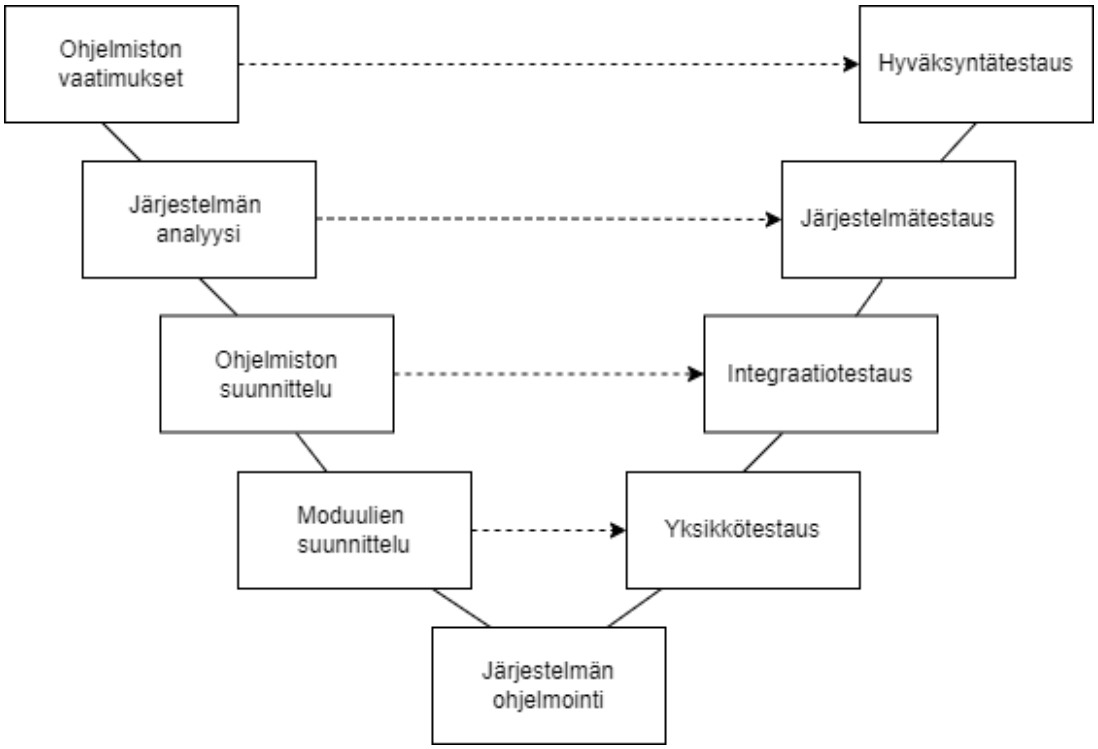
Ohjelmistotestauksessa on erilaisia tasoja, metodeja ja tyyppejä. Itse testaustekniikoita on lähes viitisenkymmentä eikä ohjelmistolle ole mahdollista, eikä järkevää käyttää kaikkia tekniikoita (Rana ym., 2019). Ohjelmistotestaustasot kategorisoidaan yleensä toiminnallisiin ja ei-toiminnallisiin testausmenetelmiin, jotka sisältävät erilaisia testausmenetelmiä.

3.3.1 Toiminnallinen testaus

Ohjelmiston toiminnallisuus on ohjelmiston ulkoista käyttäytymistä sisältäen kaikki loppukäyttäjän vaatimukset. Toiminnallinen testaus käy läpi ohjelmiston perustyönkulkua sekä vaihtoehtoisia tapauksia. Se keskittyy siihen, mitä ohjelmiston kuuluisi tehdä, eikä ota kantaa siihen, miten se tehdään. Toiminnalliset testit jakautuvat seuraaviin tasoihin: Yksikkö-, integraatio-, järjestelmä- ja hyväksyntätestaus. Jokaisessa testausvaiheessa pyritään löytämään vikoja, joita aikaisemmissa vaiheissa ei ole voitu havaita (Leung & White, 1990.) Testaustasojen järjestys voi myös vaihdella järjestelmän mukaan. Esimerkiksi hyväksyntätestaus saatetaan suorittaa ennen koko järjestelmän testausta, jos ohjelmisto on ulkopuolisen tahon kehittämä. (Hambling ym., 2010.)

Kuviossa 3 esitetään ohjelmistokehityksen V-elinkaarimalli, jossa havainnollistetaan, minkä ohjelmiston toteutusvaiheen loputtua testaustoiminnot tulisi suorittaa (Hambling ym., 2010). Ohjelmiston elinkaaressa testausvaihe on hyvin aikaa vievää, jonka vuoksi on tärkeää kattaa mahdollisimman suuri määrä koodia lyhyessä ajassa, minimimäärä testitapauksia hyödyntäen (Ramchand, Shaikh,

& Alam, 2021). V-malli havainnollistaa testauksen tarkoitusta jatkuvina toimintoina koko ohjelmiston elinkaaren ajan. (Mili & Tchier, 2015. s. 32.) Se havainnollistaa kuinka testausta voidaan suunnitella vaihe vaiheelta elinkaaren edetessä.



KUVIO 3 Ohjelmistokehityselinkaaren V-malli (Mili & Tchier, 2015)

Yksikkötestaus perustuu ohjelmistokoodin yksittäisten luokkien testaamiseen. Yksikkötestit kirjoitetaan usein samaan aikaan, kuin koodi. Yksikkötesti havaitsee usein ongelman ohjelmiston moduuleissa tai luokissa ennen kuin se integroidaan koko projektin konfiguraatioon. (Mili & Tchier, 2015, s. 24; Wagner, 2013.) Testilähtöisessä ohjelmistokehityksessä on periaatteena, että yksikkötestit kirjoitetaan ennen luokkaa, koska yksikkötestaus on samalla ohjelman ja sen haluttujen toiminnallisuuksien suunnittelua. Yksikkötestauksessa pureudutaan siis ohjelmiston koodin yksittäisiin komponentteihin ja metodeihin rivitasolla, eikä oteta kantaa ohjelmiston toimintaan kokonaisuutena. Yksikkötestauksen tarkoituksena on varmistaa, että testattava yksikkö täyttää sen toiminnalliset vaatimukset ja sen rakenne vastaa suunniteltua (Singh, 2011, s. 369). Yleisesti ohjelmistokehittäjät tekevät yksikkötestit ohjelmistokehityksen rinnalla.

Kun ohjelmiston kaikki osa-alueet on kehitetty, ohjelmisto kootaan ja testataan järjestelmävaatimusten varmistamiseksi. Integraatiotestauksessa testataan ohjelmiston yksiköiden eli moduulien vuorovaikutusta kokonaisjärjestelmän kanssa. (Mili & Tchier, 2015, s. 24.) Integraatiotestauksessa testataan moduulien vuorovaikutusta muiden komponenttien kanssa, eikä siinä oteta koodiin kantaa rivitasolla. (Leung & White, 1990.) Integraatiotestauksessa voidaan esimerkiksi varmistaa ohjelmiston vuorovaikutus tietokannan kanssa (Pittet, 2021).

Järjestelmätestaus suoritetaan, kun yksikkötestit ja integraatiotestaus on tehty. Järjestelmätestauksessa testataan koko ohjelmisto sen tarkoituksenmukaisessa ympäristössä. Koko järjestelmä on yhdistelmä ohjelmistoja, laitteistoja sekä niihin liittyviä osia, jotka yhdessä muodostavat ohjelmistotuotteen. Järjestelmätestauksessa varmistetaan, että jokainen järjestelmän toiminto toimii odotetusti, komponentit ovat yhteensopivia, ovat vuorovaikutuksessa, ja ne siirtävät dataa oikein. Järjestelmätestaus on ainut testitaso, joka testaa ohjelmiston toiminnalliset sekä ei-toiminnalliset ominaisuudet. (Singh, 2011, s. 373; Sommerville, 2016, s. 240.)

Hyväksyntätestaus suoritetaan, kun ohjelmiston katsotaan olevan valmis luovutettavaksi asiakkaalle ja se on ikään kuin jatkeena järjestelmätestaukselle (Mili & Tchier, 2015, s. 138; Singh, 2011, s. 373). Lopullisen hyväksyntätestauksen suorittaa yleensä asiakas, kun kehitystiimi on toimittanut ohjelmiston asiakkaalle testattavaksi. Asiakas saattaa kokeilla ohjelmiston toimintoja spontaanisti varmistuakseen ohjelmiston olevan sitä, mitä on tilattu (Singh, 2011, s. 373). Kuten V-malli osoittaa, hyväksyntätestaus suoritetaan perustuen ohjelmiston käyttäjävaatimuksiin (Hambling ym., 2010). On tärkeää varmistaa, että edellinen vaihe on saatu päätökseen ennen seuraavan aloittamista. Jos asiakkaan tarpeita ei ole kuvattu vaatimustenmäärittelyssä tai ne muuttuvat, ongelmat eivät välttämättä tule esiin kuin vasta hyväksyntätestauksessa. Tässä vaiheessa ongelmien korjaaminen tulee hyvin kalliiksi tai ohjelmistoprojekti voi peruuntua kokonaan. (Hambling ym., 2010.)

Testaustyypeistä puhuttaessa erityisesti tekoälyn hyödyntämisen huomioiden ei voida unohtaa regressiotestausta, joka luokitellaan ylläpitotestaukseksi. Regressiotestauksen periaate on, että kun testitapaukset on määritelty, testit voidaan ajaa aina uudelleen ohjelmistoon tehtyjen muutosten jälkeen. Tällöin voidaan varmistua, että ohjelmiston olemassa olevat toiminnallisuudet eivät ole menneet rikki muutoksen seurauksena ja se toimii vielä muokattujenkin määritysten mukaisesti. (Wagner, 2013; Leung & White, 1990.)

3.3.2 Ei-toiminnallinen testaus

Ohjelmistotestausta hyödynnetään myös ei-toiminnallisten vaatimusten laadunvarmistukseen. Ei-toiminnallinen järjestelmätestaus tarkastelee niitä toiminnallisuuksia, jotka eivät suoraan liity järjestelmän suorittamiin toimintoihin. Nämä ovat hyvin yleisiä vaatimuksia, joita voidaan soveltaa moniin eri järjestelmiin. (Hambling ym., 2010). Tässä osiossa luetellaan joitakin ei-toiminnallisia testaustekniikoita, mutta niitä ei käsitellä yhtä laajasti, kuin toiminnallisia testejä. Ei-toiminnallisia testausalueita on lukuisia kuten myös ei-toiminnallisia vaatimuksia. Tämänhetkiset tekoälyratkaisut testaukseen ovat tehty suurimmaksi osaksi toiminnalliseen testaukseen, kuten käyttöliittymän testaukseen. Tässä kappaleessa esitellään näistä yleisimmät, jotka ovat turvallisuustestaus, suorituskykytestaus, käytettävyystestaus sekä vaatimustenmukaisuustestaus.

Turvallisuustestauksessa keskitytään löytämään järjestelmän haavoittuvuudet ja suojaamaan järjestelmän data ulkopuolisilta tunkeilijoilta. Erityisesti web-sivustojen ja sovellusten tietoturvatestauksessa turvallisuustestaus

keskittyy neljään osa-alueeseen. Internet-yhteyden turvallisuudessa paneudutaan verkkoyhteyden infrastruktuurin haavoittuvuuksiin. Järjestelmän ohjelmiston turvallisuudessa arvioidaan sovelluksen toiminnan kannalta olennaisia järjestelmiä. Nämä ovat esimerkiksi käyttöjärjestelmä ja tietokantajärjestelmä. Asiakasohjelman (*eng*. *Client system*) turvallisuudessa huomioidaan, että asiakas, kuten selain, ei ole manipuloitavissa. Neljäntenä palvelinpuolen turvallisuudessa varmistetaan, että palvelimen koodi on riittävän kestävä torjuakseen tunkeutumisen. (Software Testing Fundamentals, 2020.)

Suorituskykytestaus varmistaa, että ohjelmisto vastaa syötteisiin tarpeeksi nopeasti. Nopeus tarkoittaa tässä yhteydessä ohjelmiston vasteaikaa sen työskennellessä sille asetetussa työkuormituksessa. Verkkokuormitus on yleensä suurimmillaan lounasaikaan, pörssipäivän avajaisissa tai puolen yön jälkeen. (McLeod & Everett, 2007, s. 129–130.) Lounasaikana sekä puolen yön jälkeen kuluttajilla on aikaa selata verkkokauppoja ja pörssipäivän avajaisissa osakkeiden osto on vilkkaimmillaan. Tällöin ohjelmiston tulee pystyä lataamaan sivusto tarpeeksi nopeasti, eikä kaatua kesken kaiken suuresta kuormituksesta huolimatta.

Käytettävyystestaus nimensä mukaisesti testaa sitä, onko ohjelmistoa helppo käyttää asiakkaan näkökulmasta. Helppokäyttöisyyteen sisältyy ohjelmiston ymmärrettävyys, kuinka helppoa sen käyttöä on oppia ja ymmärtää sekä onko sen käyttöliittymä houkutteleva. Viimeinen, eli vaatimustenmukaisuustestaus varmistaa, että järjestelmä täyttää sille asetetut standardit, vaatimukset tai määräykset. Testauksen taso voi vaihdella korkean tason katselmuksesta yksityiskohtaiseen standardien tarkasteluun. Tämän testaustyypin voi suorittaa myös ulkopuolinen organisaatio, jos ohjelmistoyrityksen tavoitteena on saada jonkinlainen vaatimustenmukaisuussertifikaatti tuotteelleen. (Software Testing Fundamentals, 2020.)

3.4 Testiautomaatio

Manuaalisessa testauksessa testaaja omaksuu loppukäyttäjän roolin ja suorittaa ohjelmistolle testejä testitapausten mukaisesti. Testiautomaatiossa testaajat kehittävät testikoodiskriptejä[1], jotka testaavat ohjelmiston ilman, että ihminen suorittaa testejä käyttäjän roolissa. (Amannejad & Garousi, 2014.) Kun puhutaan manuaalisesta testauksesta ja sen automatisoimisesta, manuaalisen testauksen ja testiautomaation välillä on jonkin verran vastakkainasettelua erityisesti IT-alan ajankohtaisjulkaisuissa. Näkökulma on, että manuaalisia toimintoja täytyisi pystyä automatisoimaan mahdollisimman paljon. Amannejad ja Garousi (2014) toteavat, että testiautomaatio ei aina ole kustannustehokkain ratkaisu ja testauksessa olisi tärkeää miettiä, mitkä testitapaukset kannattaa automatisoida ja mitkä tehdä manuaalisesti.

[1] Skripti tarkoittaa koodilausetta, joka suorittaa tietyn tehtävän. Termille ei löydy sopivaa suomenkielistä termiä, jonka vuoksi skripti-sanaa käytetään tässä yhteydessä.

Testiautomaation avulla ohjelmistotestausta nopeutetaan merkittävästi, sillä sen avulla voidaan ajaa iso joukko testitapauksia automaattisesti ilman ihmisen väliintuloa. Ohjelmistot kasvavat yhä laajemmiksi ja monimutkaisemmiksi, ja uusia ominaisuuksia halutaan julkaista mahdollisimman nopeasti ilman manuaalisen testauksen vaatimaa aikaa. Tämän vuoksi testiautomaatiolla on suuri merkitys ohjelmistokehityksessä. (Goericke, 2020.)

Automatisoidussa testauksessa testitapaukset määritellään testaajan toimesta manuaalisesti, mutta testien suorittaminen ja testitulosten kirjaaminen on automatisoitu. Testaaja suunnittelee ja määrittelee testitapaukset, kirjoittaa testiautomaation komentosarjat, toteuttaa ne testitapaukset, joita ei voi automatisoida ja analysoi testitulokset. (Santiago ym., 2018; King, Arbon, Santiago, Adamo, Chin, & Shanmugam, 2019.) Automaattisen testauksen ja testiautomaation ero on siinä, että automaattinen testaus on prosessi, jossa testit suoritetaan manuaalisen suorittamisen sijaan automaatiotyökaluja käyttäen. Testiautomaatiossa viitataan testien seurannan, hallinnan sekä aloitusprosessin automatisointiin. (Tricentis, 2017.) Kuitenkin monessa lähdeaineistossa testiautomaatiosta ja automaattisesta testauksesta puhutaan samana asiana.

On tärkeää kuitenkin erottaa automaation toteuttavat tekijät itse testitekijöistä. Automaatio on keino ajaa testejä, eikä se ota kantaa siihen, ovatko testit hyviä vai huonoja. (Graham & Fewster, 2012.) On myös huomioitava, että automaattisen testauksen implementoiminen, ylläpito sekä epäonnistuneiden testien analysointi voivat viedä enemmän aikaa manuaaliseen testaukseen verrattuna. Regressiotestaus on esimerkki testauksesta, joka olisi hyvä automatisoida sen toistettavuutensa vuoksi (Amannejad & Garousi 2014). Manuaalinen testaus ja testiautomaatio kulkevat rinnakkain, eivätkä ne ole toisiaan poissulkevia.

Jatkuva integrointi eli Continuous Integration (CI) on yleinen testien automatisointitapa ohjelmistokehityksessä (Graham & Fewster, 2012). Jatkuvassa integroinnissa ohjelmiston versioita testeineen julkaistaan mahdollisimman usein ja automaattisia julkaisutyökaluja käyttäen. Usein automaattinen testaus onkin osa julkaisuputkea (*eng. Deployment Pipeline*). Se on sovelluksen rakennus-, käyttöönotto-, testaus- ja julkaisuprosessin automaattisen toteutuksen työkalu. (Goericke, 2020.) Joka kerta, kun ohjelmistosta julkaistaan uusi versio, julkaisuputki huolehtii, että kaikki testit suoritetaan hyväksytysti ennen uuden version käyttöönottoa.

Yksi suosituimmista automaatiotestiteknologioista on Robot Framework. Se perustuu Python-ohjelmointikieleen ja on hyväksyntätestaukseen tarkoitettu avoimen lähdekoodin testiautomaatioympäristö. Robot Frameworkin kehitti Pekka Klärck osana diplomityötään Nokia Siemens Networkille vuonna 2005. (Bisht, 2013, s. 26.) Robot Framework ei ole riippuvainen mistään teknologiasta tai sovelluksesta, joten sen avulla voidaan testata lähes mitä vain. Testitiedot ovat taulukkomuodossa ja käynnistyttäessä se käsittelee testitiedot ja generoi lokit sekä raportit testitapauksesta. Ympäristö ei tiedä mitään testattavasta kohteesta, ja vuorovaikutus testitapauksen kanssa hoituu testikirjastojen avulla. Kirjastot voivat käyttää sovelluksen rajapintoja suoraan tai alemman tason testityökaluja ohjaiminaan. (Stresnjak & Hocenski, 2011.)

Volkin (2021) mukaan vuodesta 2020 lähtien testiautomaatioalustat ja testiautomaatioskriptit ovat olleet vahvasti ohjelmistotestaukseen liittyvän keskustelun aiheena. Ideaalitapauksessa valmista koodia ei ole olemassa ilman siihen tehtyjä testejä versionhallinnassa. Todellisuus voi kuitenkin näyttäytyä toisenlaisena, sillä testauskäytännöissä sallitaan yhä suppeampia testikattavuuksia. Testaustiimit kirjoittavat jatkuvasti lisää koodia ja päivityksiä useista hajautetuista mikropalveluista koostuviin sovelluksiin, jolloin testaus on usein osa-alue, josta tingitään ensimmäisenä. (Volk, 2021.) Testiautomaatio nykyisellään ei kykene skaalautumaan ohjelmistojen välillä eikä se pysty jäljittelemään ihmisälykkyyttä. Tästä syystä nykyisen testiautomaation ja täysin automatisoidun ohjelmistotestausratkaisun välillä on suuri kuilu. (Santiago ym., 2018.)

3.5 Ohjelmistorobotiikka

Testiautomaatiosta puhuttaessa törmätään usein myös ohjelmistorobotiikkaan, jonka avulla voidaan myös minimoida manuaalista työtä ohjelmistoihin liittyen. Ohjelmistorobotiikka (*eng. Robotic Process Automation,* RPA) on teknologia, jonka avulla automatisoidaan rutiinimaisia prosesseja. Se on ohjelmoitu niin, että se voi suorittaa sääntöpohjaisia, toistuvia digitaalisia tehtäviä nopeasti ja tehokkaasti. Yleisimpiä esimerkkejä tästä on toistuva tiedon syöttäminen järjestelmään, tiedon kopiointi ja liittäminen tai samojen tietojen täyttäminen moneen paikkaan. (Wood, 2021.) Ramchandin ym. (2021) tutkimuksen mukaan ohjelmistorobotiikkaa hyödynnettiin varsin laajasti ohjelmiston laadunvarmistuksessa. Heidän teettämänsä kyselyn mukaan 21.1 % testausammattilaisista käyttivät ohjelmistorobotiikkaa osana ohjelmiston laadunvarmistusta. Ramchand ym. (2021) määrittelee ohjelmistorobotiikan olevan käytetty tekoälyalgoritmi.

Olennaisin ero testiautomaation ja ohjelmistorobotiikan välillä on, että testiautomaatiota sovelletaan ohjelmistotuotteen ominaisuuksien testaamiseen, kun taas ohjelmistorobotiikassa automatisoidaan myös liiketoimintaprosesseja. Ohjelmistorobotiikan työkalut eivät vaadi välttämättä ollenkaan koodausta ja se toimii reaaliaikaisesti. (JavaTpoint, 2021.) Näin ollen ohjelmistorobotiikkaa sovelletaan testiautomaatiota laajemmin erityisesti hallinnollisten tehtävien automatisoinnissa, jotka vaativat ihmiseltä aikaa ja manuaalista työtä. Näitä ovat mm. laskutus, raportointi ja henkilöstöhallinnolliset toimenpiteet. Ohjelmistorobotiikkaa voivat siis hyödyntää muutkin organisaation työntekijät, kuin testausammattilaiset.

4 TEKOÄLYN HYÖDYNTÄMINEN OHJELMISTOJEN LAADUNVARMISTUKSESSA

Kappaleessa kuvaillaan, mitä tekoäly ohjelmistojen testauksessa on tänä päivänä, mitä siihen liittyy ja millaisia vaikutuksia sillä tiettävästi on. Kappaleessa käydään läpi myös joitakin olemassa olevia teknologioita ja yrityksiä, joissa tekoälyä ja koneoppimista on käytetty ohjelmistotestaukseen. Koska kyseessä on kehittyvä alue tekoälyn soveltamiskentässä, kappaleessa pohditaan myös tekoälyn soveltamisen haasteita ohjelmistotestauksessa.

4.1 Tekoälypohjaiset testausteknologiat

Tekoälyn tarkoitus ohjelmistotestauksessa on hyödyntää tekoälyalgoritmeja ohjelmistotuotteen testaukseen. Tekoälyn avulla testausprosessista pyritään saamaan älykkäämpää ja tehokkaampaa. (TestingXperts, 2022). Tämä voi tarkoittaa ohjelmistoa, joka pystyy testaamaan itse itseään tai parantamaan toimintaansa itsenäisesti (King ym., 2019). Valmiita tekoälypohjaisia testausteknologioita on jo olemassa jonkin verran. Erityisesti suuret teknologiayhtiöt ovat omaksuneet tekoälyn osaksi ohjelmistokehitystä. Malviya (2020) toteaa, että nykyään organisaatioilla alkaa olla käytössään jo parempia koneoppimisalgoritmeja erilaisten testausmallien analysointiin ja suurien tietomäärien käsittelyyn. Tämä voi johtaa parempiin ajonaikaisiin testauspäätöksiin. (Malviya, 2020.) King ym. (2019) konferenssin paneelikeskustelussa tekoälypohjaiset testausmenetelmät kategorisoitiin seuraavasti:

- **Differentiaaliset menetelmät:** Sovellusversioita verrataan sen rakenteisiin, niiden erot luokitellaan ja luokittelun perusteella saatua palautetta hyödynnetään oppimiseen.
- **Visuaaliset menetelmät:** Hyödynnetään kuvatunnistusta vertailemalla kuvakaappauksia käyttöliittymien ulkoasun testaamiseksi.

- **Deklaratiivinen menetelmä:** Testin tarkoitus määritellään luonnollisella kielellä ja pyydetään määrittelijää selvittämään, miten testi suoritetaan.
- **Itseohjautuva menetelmä:** Testit ylläpitävät itseään automaattisesti käyttöliittymän muuttuessa.

Tällä hetkellä saatavilla olevista tekoälytestausteknologioista visuaalinen testaus on saanut eniten painoarvoa. Visuaalisen testauksen avulla testaustiimillä on ylimääräinen silmäpari käyttöliittymän tarkastukseen, jolloin tiimi voi keskittyä ohjelmiston osa-alueisiin, joiden kehittämiseen tarvitaan todellista ihmisen älykkyyttä. (Volk, 2021.) Visuaalisessa testauksessa kuvaperusteiset algoritmit voidaan kouluttaa analysoimaan käyttöliittymiä ja lisäämään testausprosesseja, jotta käyttöliittymän näkymän oikeellisuus voidaan varmistaa. Esimerkiksi DOM-elementtien (*eng. Domain Object Model*) tutkimiseen kirjoitetun koodin sijaan tekoäly analysoi käyttöliittymän tuotosta odotettuun tuotokseen kuvakaappausta hyödyntämällä. DOM-elementti on olio, joka sisältää tiedon, miten HTML/XHTML/XML-komponentit on muodostettu käyttöliittymässä. Jos tuotos vastaa kuvakaappausta, testi menee läpi. (Fogg, 2021.) Applitools on esimerkki testausteknologiasta, jossa hyödynnetään visuaalista tekoälyä testitapausten luonnissa. Sen tekoäly jäljittelee ihmisen näköä ja aivoja havaitakseen toiminnallisia ja visuaalisia regressioita käyttöliittymän testauksessa. (Applitools, 2021.)

Tekoälypohjaisessa koodittomassa testimenetelmässä on tarkoituksena luoda itse itseään parantelevia testitapauksia, jotka eivät vaadi testien ylläpitoa ihmisen toimesta, kuten testiautomaatiossa. Foggin (2021) mukaan tekoäly pystyy argumentoimaan tallennus- ja toistoprosessia luomalla dynaamisesti olioiden sijaintitietoja, kun niitä käytetään. Käyttäjän antamat komennot ja objektityypit tunnistetaan, oli kyseessä sitten pudotusvalikko tai syötekenttä. Tällä tavalla käyttöliittymätestauksessa säästetään runsaasti aikaa. (Fogg, 2021.)

Organisaatiot ovat kehittäneet valmiita tekoälyyn pohjautuvia testausmenetelmiä. Netflix on kehittänyt Lerner-nimisen testausjärjestelmän, joka koostuu mikropalveluista, Python-kirjastosta ja skaalautuvista testiagenteista. Järjestelmä valitsee ja priorisoi testitapauksia hyödyntämällä tekoälyssä vahvistusoppimista testiajojen keston, edellisen ja viimeisen testiajon sekä epäonnistumisen mukaan. Lerner ehdottaa testitapauksia, joiden perusteella ohjelmistosta pystytään löytämään virheitä nopeammin. Lernerin testausrajapinnat on jaoteltu kolmeen alueeseen: testisuoritustulosten tallennus, testiympäristön tilan perusteella saadut testisuositukset sekä testiagentin palkitseminen perustuen testien suoritustulokseen. (Kirdey, Cureton, Rick, Ramanathan & Shukla, 2019.)

Lerner on hyödyntänyt Spiekerin, Gotliebin, Marijan ja Mossigen (2017) esittelemää vahvistusoppimisen menetelmää. Menetelmässä koneoppimisalgoritmi oppii itsenäisesti suoritusympäristönsä kokemuksesta hyödyntäen vahvistusoppimista. Menetelmä pyrkii valitsemaan ja priorisoimaan testitapaukset neuroverkkoa käyttämällä valitussa testitapauksissa ja niiden suoritusympäristössä. Priorisoituja testitapauksia on käytetty onnistuneesti vikojen

havaitsemiseen aiemmissa jatkuvan integraation sykleissä. Suoritusjärjestys asetetaan suorittamalla lupaavimmat tapaukset ensin. (Spieker ym., 2017.) Santiago ym., (2018) puhuvat myös vahvistusoppimisen puolesta järjestelmän vikojen etsintäprosessissa. Sitä voidaan käyttää älykkään järjestelmän palkitsemiseen järjestelmän kaatumisen tai muun vian paljastamisesta.

Facebook on kehittänyt koneoppimisjärjestelmän regressiotestaukseen. Sen avulla luodaan todennäköisyysmalli regressiotestien valitsemiseksi koodimuutoksia varten. Koneoppimisjärjestelmä kehittää automaattisesti testivalintasuunnitelman oppimalla laajasta koodimuutosten ja testitulosten joukosta. Tämän harjoitusdatan avulla koneoppimisjärjestelmä oppii mallin, joka perustuu aiemmista koodimuutoksista ja testeistä johdettuihin ominaisuuksiin. Kun järjestelmä analysoi uusia koodimuutoksia, opittua mallia käytetään luomaan abstraktio koodimuutoksesta, jolloin malli kykenee ennustamaan regression havaitsemisen todennäköisyyden minkä tahansa testin osalta. Malli oppii tuloksista ja ajan myötä paljastaa myös virheet nopeammin. (Machalica, Samylkin, Porth & Chandra, 2018.) Test.ai on ollut yksi vaikuttajista valmiisiin tekoälypohjaisiin testausteknologioihin. Sen järjestelmä pystyy käymään läpi monen mobiilisovelluksen tiedot samanaikaisesti ja käyttämään uudelleen yleisiä testitapauksia näissä sovelluksissa. Sen teknologiassa hyödynnetään vahvistusoppimista ja konenäköä testitapausten luontiin. (Santiago ym., 2018; Test.ai, 2022.)

Functionize on yhdysvaltalainen, toiminnallisiin testeihin erikoistunut testausalusta, joka hyödyntää koneoppimismalleja. Sen avulla voidaan rakentaa testejä nopeammin ja vähentää niiden ylläpitoa. Testit luodaan käyttäen Functionizen arkkitehti- tai luonnollisen kielen prosessointitoimintoa. Arkkitehtia käytetään testitapausten luomiseen käymällä läpi sivuston työnkulku valitsemalla arkkitehdin toimintolistasta elementtipohjaisia tai ei-elementtipohjaisia toimintoja testille. Nämä ovat esimerkiksi rajapintakutsu, klikkaus, tietokantayhteyden tarkastus tai syötekenttä. Se toimii paremmin, mitä enemmän sillä on käsiteltävänä dataa, jota se käyttää vuorovaikutuksessa käyttöliittymän kanssa.

Luonnollisen kielen toiminnossa yksittäinen testi luodaan luonnollisella kielellä kirjoittaen. Testissä tulee kuvailla testitapauksen vaiheittaista työnkulkua, joka luodaan käyttämällä NLP-prosessia avainsanalähtöisen komentosarjan kanssa. Functionizessa hyödynnetään myös konenäköä. Alusta luo visuaalisen mallin testattavasta sovelluksesta käyttämällä testien luomisen aikana kerättyjä tietoja. Myöhempiä testiajoja verrataan sovelluksen lähtötasoon, jotta voidaan tunnistaa virheet tai poikkeamat. Konenäkötoiminnon avulla voidaan verrata visuaalisesti elementtejä tai kokonaisia sivuja. (Functionize, 2022.)

Edellä mainitut teknologiat hyödyntävät tekoälyn alalajeista koneoppimista ja erityisesti vahvistusoppimista, neuroverkkoja, luonnollisen kielen prosessointia sekä konenäköä. Houranin ym. (2019) mukaan luonnollisen kielen prosessointia pystytään hyödyntämään testitapausten priorisointiin, manuaalisen testitapauksen tulosten ennustamiseen, ohjelmiston vaatimuksiin perustuvien testitapausten generointiin, yksikkötestien dokumentointiin sekä päällekkäisten virheraporttien havainnointiin. Esimerkiksi Functionizessa luonnollisen kielen prosessointia käytetään testitapausten luomiseen.

Malviya (2020) korostaa syväoppimismenetelmien tärkeyttä, jotta tulevaisuuden autonominen testaus tullaan saavuttamaan. Esimerkiksi itseajavissa autoissa autonomiset teknologiat generoivat itse niiden omat skriptit oppimalla järjestelmänsä kautta. Tällä tavalla järjestelmä pystyisi itse oppimaan ja testaamaan sovelluksia ilman dokumentoituja testitapauksia tai ihmisen kirjoittamaa koodia. (Malviya, 2020.) Santiago ym (2018) toteavat samat seikat luonnollisen kielen prosessoinnin ja vahvistusoppimisen käyttökohteista.

4.2 Tekoälyn hyödyt ohjelmistotestauksessa

Monissa IT-alan ajankohtaisjulkaisuissa luonnehditaan, että tekoälyavusteisen ohjelmistokehityksen tavoitteena on, että tekoälyn hoidettavaksi annetaan manuaaliset, puuduttavat ja toistoa vaativat rutiinitehtävät, ja testaaja käyttää aikaansa vaativampiin kokonaisuuksiin, johon tekoäly ei vielä taivu. Tutkimusyhtiö Gartnerin mukaan tekoälytehosteinen ohjelmointi saavuttaa lakipisteensä hypekäyrällä 5–10 vuoden kuluttua. Santiago ym. (2018) jakaa saman käsityksen. Tällöin teknologiat leviävät markkinoille ja jalkautuvat käytännön työhön. (Virtanen, 2021.)

Kingin ym. (2019) kyselytutkimuksessa 23 % vastaajista 328 vastaajan joukosta oli sitä mieltä, että tekoälypohjainen testaus tulee korvaamaan manuaalisen testauksen noin 5 vuoden sisällä, kun taas 35 % oli sitä mieltä, ettei tekoäly tule korvaamaan koskaan täysin manuaalista testausta. Loput vastaukset jakautuivat 10–20 vuoden välille. Kuitenkin 42 % vastaajista uskoi tekoälyn vaikuttavan manuaaliseen testaukseen jo vuoden 2020 aikana. Kysely toteutettiin vuoden 2017 aikana ohjelmistoalan asiantuntijoille, jotka olivat kiinnostuneita tekoälyn soveltamista testaukseen ja/tai tekevät parhaillaan työtä tekoälytestauksen parissa. (King ym., 2019.) Tekoälykeskustelussa puhutaan paljon manuaalisen testauksen korvaamisesta. Testiautomaation tarkoituksena on hoitaa toistettavien ja laajojen testitapausten ajaminen, joten herää kysymys, mihin tekoälyä lopun viimein tarvitaan. Testiautomaatio jo tällä hetkellä hoitaa paljon manuaalista, toistettavaa testausta, joten tähän osa-alueeseen ei välttämättä tekoälyä tarvita.

Houranin ym. (2019) mukaan autonominen testaus tulee säästämään aikaa ja parantamaan testauksen tarkkuutta. Autonomisella testauksella tarkoitetaan tässä tapauksessa automaattisesti generoituvia testitapauksia ja niiden ajamista automaattisesti. Bellapu (2021) toteaa, että manuaalisessa testauksessa testattavia toiminnallisuuksia on paljon, mikä tarkoittaa lukuisien koodirivien luontia. Jos asiakas löytää ohjelmistosta virheitä kehitysvaiheen loppuvaiheesta, kustannukset kasvavat merkittävästi. Tällaisilta skenaarioilta voidaan Bellapun (2021) mukaan välttyä, jos tekoäly onnistuu keventämään merkittävästi testaustyön kuormittavuutta ja sen avulla voidaan saavuttaa suurempi testauskattavuus nopeammin. Hourani ym. (2019) nostaa esiin tekoälytestauksen avulla odotetut edut Kuviossa 4.

KUVIO 4 Tekoälyn hyödyt ohjelmistotestauksessa (Hourani ym., 2019)

Kuvion mukaan tekoäly oletettavasti käsittelee nopeammin ja tarkemmin suurta datamäärää ja on toiminnoissaan sekä tuloksissaan ihmistä tarkempi. Fogg (2021) luettelee seuraavia tekoälyn hyödyntämismenetelmiä testauksessa: automaattinen testisarjan luominen annetuilla testitiedoilla, ohjelmiston tulosten analysointi visuaalisesti ja ohjelmiston virheiden havaitseminen, joita ei löydetä perinteisillä testeillä (Fogg, 2021). Ramchand (2021) lisää vielä edellä mainittuihin hyödyntämismenetelmiin maksimaalisen testikattavuuden saavuttamisen, testitapausten kokonaismäärän vähentämisen ja testitapausten suodattamisen. Näiden menetelmien tulisi tehostaa ja keventää ohjelmistotestaustyötä.

Volk (2021) perustelee vielä, että tekoälytehosteinen testaus voi lisätä ohjelmiston laadunvarmistuksen tehokkuutta muun muassa karsimalla pois testitapaukset, jotka eivät vaadi ihmisen huomiota, automatisoimalla kasvavaa osuutta testauksen kokonaistyönkulusta, yhdistämällä inhimillisiä tehtäviä tehokkuuden lisäämiseksi, tarjoamalla käyttökelpoisia suosituksia testitiimille ja oppimalla ihmisen tekemistä päätöksistä. Tekoälytestaus voisi parhaimmillaan poistaa organisaatioiden paineen nopean julkaisun, kustannusten alentamisen ja lopputuotteen laadun parantamisen hankalasta kierteestä. (Volk, 2021.) Näiden oletettujen etujen perusteella tekoäly voisi löytää ohjelmistosta enemmän testattavaa ja priorisoimaan testit sen perusteella, mihin kannattaa ihmisen kiinnittää huomiota. Lisäksi tekoäly voisi myös analysoida testituloksia ja oppia niistä, jolloin testaajan ei tarvitsisi käyttää aikaa testitapausten luokitteluun ja analysointiin.

Vaikka ohjelmistotestaus on pitkälti dynaamista testien ajoa, myös staattiset testausmenetelmät, kuten koodikatselmukset ja koodin tutkiminen eri näkökulmista, voidaan lukea ohjelmistotestaukseksi. Tekoälyä voidaan mahdollisesti hyödyntää myös näihin osa-alueisiin osana kokonaisvaltaista ohjelmiston laadunvarmistusta. (Tieturi Oy, 2021.) Tällaiseen testaustapaan testiautomaatio ei

kykene. Ohjelmiston ja koodin tutkiminen vaatii asioiden analysointia eri näkökulmista ja päätösten tekoa tiedon perusteella, joka voidaan tulkita älykkääksi toiminnaksi. Tällaiset tapaukset ovat oiva kohde hyödyntää tekoälyä.

4.3 Haasteet tekoälypohjaisessa testauksessa

Tämänhetkiset tekoälypohjaiset testausmenetelmät perustuvat suurilta osin koneoppimiseen, joka kehittyy oppimisdatan myötä paremmaksi ja tarkemmaksi. Yhtenä haasteena on, milloin voidaan varmistua siitä, että testiohjelmiston oppimisdata on vakiintunutta ja testausohjelmisto generoi oikeanlaiset testit. Spiekerin ym. (2017) vahvistusoppimistapauksessa lähdettiin täysin mallittomasta testitapauspriorisoinnista liikkeelle. Noin 60 jatkuvan integrointisyklin jälkeen, järjestelmä oppi tehokkaasti priorisoimaan testejä. Jotta koneoppiminen saadaan koulutettua tekemään oikeanlaisia päätöksiä ja keventämään manuaalista kuormaa, se vaatii työtä.

Koneoppimismallissa testausalgoritmi oppii vain sen historiadatan perusteella. Koneoppimisessa ohjelmisto ei kykene välttämättä ratkaisemaan sille vieraita tapauksia ja ongelmia itsenäisesti, jolloin testit voivat jättää huomiotta joitakin tapauksia. Neittaanmäki ja Tuominen (2019) toteavat, että nykyiset tekoälyalgoritmit eivät voi lisätä tai luoda oppimaansa malliin tietoa, joia niiden opettamiseen käytettävässä datassa ei ole. Tekoälyn laatu ja kattavuus on datasta riippuvaista, jolloin ihmiselle helppojen tehtävien suorittaminen on tekoälylle haastavaa, jos data on epäformaalissa muodossa. (Neittaanmäki & Tuominen, 2019.) Testausteknologiat eivät siis kykene vielä täysin jäljittelemään ihmistestaajan älykkyyttä ja luovuutta, vaikka joissakin julkaisuissa puhutaankin tekoälyn korvaavan ihmistestaajat tulevaisuudessa.

Santiago ja muiden (2018) mukaan testausmaailmassa on tehty viime aikoina paljon tutkimuksia puoliautomatisoiduista testiteknologioista, kuten mallipohjaisesta testien generoinnista. Nämä teknologiat eivät kuitenkaan jäljittele ihmistestaajan ajattelua ja oppimista eivätkä skaalaudu saumattomasti sovellusten ja sovellusalueiden välillä. Santiagon ym. (2018) mukaan testausyhteisö on jäljessä tämänkaltaisissa innovaatioissa, sillä testaukseen on tarve rakentaa älykkäämpiä ratkaisuja, jotka voisivat toimia yhtä älykkäästi kuten itseajavat autot ja muut vastaavat älykkäät järjestelmät. (Santiago ym., 2018.)

Tästä päästään pohdintaan siitä, miten älykkäitä ratkaisuja saadaan kehitettyä testaukseen. Tekoälyn ja koneoppimisen hyödyntäminen ohjelmistojen laadunvarmistuksessa vaatii oikeanlaista osaamista. Testiautomaation parantaminen ja luominen tekoälyn avulla vaatii laajaa teknistä, matemaattista ja tilastollista osaamista yhdistettynä rohkeuteen ja luovuuteen (QA Madness, 2021.) Ramchand ym. (2021) on myös sitä mieltä, että laadunvarmistusasiantuntijoilla ei ole tarpeeksi taitoja tekoälyn hyödyntämiseen. Hänen mielestään organisaatioiden tulisi kouluttaa testausammattilaisia ymmärtämään tekoälyyn liittyviä kokonaisuuksia, kuten tekoälytestaus, matemaattinen optimointi, liiketoimintaäly ja algoritmien analysointi. (Ramchand ym., 2021.)

Ohjelmistotestaus automatisoiduilla työkaluilla ei myöskään poista vielä manuaalista testausta kokonaan, eikä se täten korvaa välttämättä ihmisen toteuttamaa testausta. Testaajan tulee edelleen kirjoittaa koodi kuten ennenkin kohteen toimivuuden testaamiseksi. Kun testi on todettu toimivaksi, koodia voidaan käyttää uudelleen muutosten testaukseen, jolloin testauskoodin uudelleenkirjoittaminen eliminoituu. (Bellapu, 2021.) Testitiimin täytyy myös valvoa ja varmistua siitä, että tekoälyjärjestelmä toimii oikein, eikä se jätä huomiotta vakavia ohjelmiston virheitä, jotka voivat näyttäytyä asiakkaalle myöhemmissä vaiheissa. Tästä syystä olisi tärkeää hyödyntää Rudinin (2019) mukaan tulkittavia tekoälyoppimismalleja mustalaatikkomallien sijaan. Tällöin virheellisiin testausalgoritmeihin voidaan puuttua ja testaaja ymmärtää, mihin perustuen tekoäly generoi ohjelmiston testitapaukset.

Kappaleessa on käsitelty, mitä tekoälytestaus on tällä hetkellä sekä mitkä ovat sen mahdollisuudet ja haasteet. Teoria-aineiston perusteella voidaan todeta, että tekoälyavusteisia teknologioita on jo valmiina jonkun verran. Visuaalinen käyttöliittymätestaus, testitapausten priorisointi ja testitapausten luominen ovat osa-alueita, joihin tekoälyä voidaan testauksessa mahdollisesti hyödyntää. Näiden osa-alueiden odotetaan keventävän testaajien manuaalisen työn kuormaa, johon testiautomaatio ei kykene. Tutkimuksen seuraavassa osiossa siirrytään tutkimuksen empiiriseen osioon ja haetaan vastauksia teoria-aineistosta nousseisiin kokonaisuuksiin.

5 TUTKIMUKSEN TOTEUTUS

Tässä kappaleessa esitellään tutkimuksen empiirinen osio sekä käydään läpi valitut tutkimusmenetelmät. Ensin esitellään tutkimuksen tavoite ja tutkimuskysymykset. Seuraavaksi perustellaan tutkimusmenetelmän valinta, kuvataan aineistonkeruu ja lopuksi käsitellään tutkimuksen toteutus.

5.1 Tutkimuksen tavoite

Tutkimuksen tavoitteena on selvittää, miten tekoälyä hyödynnetään ohjelmistojen laadunvarmistustyössä. Tarkoitus on saada tietoa tekoälypohjaisen ohjelmistotestauksen- ja laadunhallinnan nykytilasta Suomessa toimivissa yrityksissä. Lisäksi halutaan tietää, onko tekoäly ratkaissut nykyisiä ohjelmistotestaukseen liittyviä haasteita ja onko se helpottanut manuaalisesta testauksesta aiheutunutta työkuormaa. Jotta nämä tutkimuksen tavoitteet toteutuvat, tutkimuskysymykset ovat muotoutuneet seuraavasti:

- Miten tekoälyä hyödynnetään ohjelmistotestauksessa IT-organisaatioissa Suomessa?
- Millä tavalla tekoäly on muuttanut testausta organisaatiossa?

On yleistä, että tutkimuskysymys muuttuu ja tarkentuu tutkimusprosessin kuluessa. Tästä huolimatta alustavakin tutkimuskysymys on tutkimusprosessin arvokkain resurssi ja sen puute on tutkimuksen sujuvan etenemisen suurin hidaste. (Eriksson & Koistinen, 2014.) Tekoälyteknologiat ohjelmistotestauksessa eivät vielä ole saavuttaneet tekoälymääritelmän mukaista ihmisälykkyyttä, mutta koneoppimisen saralla edistystä testauksen automatisointiin on tullut. Aihetta käsittelevien ajankohtaisjulkaisujen ja yleisen keskustelun perusteella ilmiö tulee suuntautumaan siihen, että tekoäly tulee korvaamaan manuaalisen ja puuduttavan testaustyön tulevaisuudessa ja saavuttamaan mahdollisesti ihmistestaajan ongelmanratkaisukyvyn.

Tutkimuskysymyksillä haluttiin selvittää, onko tämä paljon esillä ollut aihe jalkautunut jo organisaatioiden käytäntöön Suomessa. Lisäksi haluttiin laajentaa ymmärrystä siitä, millaista tekoälypohjainen ohjelmistotestaus on käytännössä. Tutkimuksen empiirisessä osiossa pyrittiin löytämään yrityksiä, jotka ovat jollain tavalla hyödyntäneet tekoälyä ja koneoppimista ohjelmistotestauksessa. Vaikka tutkimuksessa puhutaan tekoälystä ja koneoppimisesta erillisinä käsitteinä, empiiriseen osiossa koneoppiminen sisällytetään tekoälyn piiriin.

5.2 Tutkimusmenetelmä

Laadullinen tutkimus tarkoittaa sellaista tutkimusta, jossa tutkitaan ilmiötä ilman tilastollisia menetelmiä tai määrällisiä keinoja. Laadullisessa tutkimuksessa käytetään sanoja ja lauseita, eikä se perustu lukuihin. Tarkoituksena on ilmiön kuvaaminen, ymmärtäminen ja tulkitseminen. Laadullinen tutkimus on usein kuvailevaa, jossa tutkija on kiinnostunut prosesseista, merkityksistä sekä ilmiön ymmärtämisestä sanojen, tekstien ja kuvien avulla. (Kananen, 2008, s. 24.)

Tutkimusmenetelmäksi on valikoitunut kvalitatiivinen monitapaustutkimus. Tapaustutkimus on tutkimusmenetelmä, jossa perehdytään yksittäisten tapausten dynamiikkaan, eli siinä pitäydytään vain muutamassa tai vain yhdessä havaintoyksikössä. Tapaustutkimus voi sisältää yhden tai useamman tapauksen ja niiden määrittely, analysointi ja ratkaisu on tapaustutkimuksen tavoite. Tapaus voi olla yksilö, ryhmä, ohjelma tai prosessi tai ilmiö. Oleellista tutkittavalle tapaukselle on, että sen pystyy rajaamaan selkeästi muusta kontekstista. Tapauksen määrittely voidaan tehdä ennen aineiston keruuta tai sen jälkeen. (Eisenhart, 1989; Eriksson & Koistinen, 2014; Hirsjärvi & Hurme, 2008, s. 58.) Eisenhardtin (1989) mukaan usean tapauksen tutkimukset ovat hyödyllisiä silloin, kun pyritään luomaan uutta teoriaa, koska useampi tapaus mahdollistaa replikoinnin. Replikoinnissa voidaan yksittäisten tapausten avulla vahvistaa joko etukäteen tai tutkimuksen kuluessa määriteltyjä teoreettisia propositioita, eli väitelauseita.

Tärkein tapausten valintaan vaikuttava asia pitäisi olla kysymys: ”Mitä voimme oppia tästä tapauksesta?” (Eriksson & Koistinen, 2014.) Tutkimuksen tapaukset hyödyntävät jollakin tavalla koneoppimista tai tekoälyä ohjelmistotestauksessa. Tekijä arvioi, että suuret organisaatiot ovat uusimpien teknologioiden omaksumisessa pienempiä edellä johtuen isommista resursseista. Tämä tarkoittaa rahallista panostusta uusimpiin teknologioihin sekä parhaan osaamisen palkkaamista. Lisäksi isoimmat organisaatiot pystyvät ratkomaan isojen ja vaativien asiakkaiden ongelmia, jolloin parhaat työkalut tulee olla käytettävissä. Pienemmillä organisaatioilla löytyy kuitenkin tarvittaessa enemmän ketteryyttä ja reagointikykyä muutoksiin, kuin suurilla organisaatioilla.

Haastateltavia päätettiin lähteä kartoittamaan ensisijaisesti isommista yrityksistä, liikevaihtoluokaltaan kymmenen miljoonan euron molemmin puolin. Organisaation koko ei kuitenkaan ollut määräävä tekijä haastateltavien etsimisessä, vaan haastateltavan asiantuntijuus ja kokemus alalta. Tekijä perehtyi alan ajankohtaisjulkaisuihin ja haastateltaviksi etsittiin henkilöitä, jotka ovat

käyttäneet puheenvuoroja ohjelmiston laadunhallintaan ja tekoälyyn liittyen. Haastateltavien etsintään käytettiin muun muassa Tivi-lehteä, Mimmitkoodaa-ohjelman järjestämää virtuaalitapahtumaa, Jyväskylän yliopiston DuunIT-messuja, LinkedIniä ja Googlen hakukonetta.

Eisenhardtin (1989) mukaan tapausten ideaalilukumäärää tutkimusta kohden ei ole olemassa. Kuitenkin hän perustelee, että 4–10 tapausta on toiminut hyvin tutkimuksissa, sillä alle neljän tapauksen avulla on vaikeaa tuottaa propositioita ja yli kymmenen tapauksen kohdalla aineistonkäsittely aineiston laajuuden vuoksi voi osoittautua tutkijalle vaikeaksi. Yhtenä haasteena tapausten etsimisessä oli, että ennakkoselvityksen perusteella yrityksillä on kiinnostusta tekoälyn mahdollisuuksiin testauksessa, muttei käsitystä, miten lähteä hyödyntämään sitä. Kartoituksessa tuli myös ilmi, että läheskään kaikilla kontaktoiduilla yrityksillä ei ole tekoälyavusteisesta testauksesta kokemusta. Tapaukset rajattiin sen mukaan, onko yritys tehnyt jo jonkinlaista kokeilua ohjelmiston laadunhallintaan hyödyntäen koneoppimista, vaikka se ei laajassa käytössä vielä olisikaan. Tällä varmistettiin se, että tutkimukseen löydetään tapauksia, sillä tiukempi rajaus olisi voinut jättää tutkimuksen otannan liian suppeaksi.

5.3 Aineistonkeruu

Tapaustutkimuksessa empiirisen aineiston keruu toteutetaan usein yhdistämällä menetelmiä kuten arkistointi, haastattelut, kyselyt ja havainnointi. Menetelmä voi olla kvalitatiivinen, kvantitatiivinen tai molempia. Tapaustutkimusmenetelmä sopii moneen tarkoitukseen, kuten ilmiön kuvaamiseen, teorian testaamiseen ja luomiseen. (Eisenhart, 1989.)

Tutkimuksen empiirinen osio toteutetaan kvalitatiivisella teemahaastattelulla. Kvalitatiivinen teemahaastattelu on yleisin ja tärkein tiedonkeruumenetelmä kvalitatiivisessa tutkimuksessa. Teemahaastattelussa tyypillisesti haastattelun aihepiirit eli teemat ovat etukäteen tiedossa, mutta kysymykset eivät ole tarkasti järjestelty tai muotoiltu. Yksityiskohtaisten kysymysten sijaan haastattelu etenee tutkimuksen teemojen varassa. Teemahaastattelun etuna muihin tiedonkeruumenetelmiin verrattuna on sen joustavuus aineiston keruussa. Aineiston keruuta voi teemahaastattelun avulla säädellä tilanteen edellyttämällä tavalla myötäillen haastateltavia. (Hirsjärvi, Remes & Sajavaara, 2003, s.194–197; Hirsjärvi & Hurme, 2008, s. 48.)

Joustavuutensa vuoksi teemahaastattelu sopii moniin tutkimustarkoituksiin. Kielellisen ja suoran vuorovaikutustilanteen ansiosta, teemahaastattelutilanne luo mahdollisuuden ohjata tiedonhankintaa haastattelutilanteen aikana. Tällöin on mahdollista saada selville haastateltavalta vastausten taustalla olevia motiiveja. (Hirsjärvi & Hurme, 2008, s. 34.) Teemahaastattelu toteutettiin tutkimuksessa puolistrukturoituna. Puolistrukturoidussa haastattelussa tutkija on valmistellut joitakin haastattelukysymyksiä etukäteen, mutta spontaanille keskustelulle on haastattelutilanteessa jätetty tilaa. (Myers & Newman, 2007.)

Tutkimuksen tarkoituksena on kartoittaa tekoälyavusteisen ohjelmiston laadunvarmistuksen nykytilaa IT-organisaatioissa Suomessa. Tämä edellyttää tiedon saamista organisaatiossa työskentelevältä henkilöltä, jolloin tutkimuksen aineistonkeruun tulee palvella tätä tarkoitusta. Teemahaastattelu aineistonkeruumenetelmänä sopii parhaiten tutkimuksen toteutukseen, sillä organisaatioiden testausammattilaisilla on tutkimuksen teemoista paras tieto ja kokemus. Näitä kokemuksia haluttiin kuulla ja hyödyntää tutkimuksessa. Kuten Hirsjärvi ja Hurme (2008, s. 105) toteavat, teemahaastattelukysymykset tulisivat edistää myönteistä vuorovaikutusta, ylläpitää keskustelua ja motivoida haastateltavia puhumaan kokemuksistaan. Tätä tutkimuksen aineistonkeruumenetelmällä tavoitellaan.

Tutkimusaihe on myös suhteellisen tuore. Teemahaastattelu sopii tutkimusaiheeseen, joka on vähän kartoitettu. Tutkijan on täten vaikea tietää haastateltavan vastausten suuntia. (Hirsjärvi & Hurme, 2008, s. 35.) Puolistrukturoitu teemahaastattelu tiedonkeruumenetelmänä antaa haastateltaville mahdollisuuden kertoa kokemuksistaan tekoälyavusteisesta testauksesta vapaasti, jolloin tietoa ilmiön todellisista hyödyistä ja haasteista saadaan monipuolisesti. Tutkimuksessa halutaan tietää käytettyjen tekoälyratkaisujen lisäksi, ovatko ne vähentäneet manuaaliseen testaukseen käytettävää aikaa.

Tutkimuksen aihe muotoutui syyskuussa 2021 tutkimuksen ohjaajan kanssa keskustellessa tutkimuksen teemoista ja tutkijan omista mielenkiinnon kohteista. Tekijä on aikaisemmassa työssään ollut paljon tekemisissä erilaisissa laadunvarmistusprojekteissa, joten ohjelmistojen ja tuotteiden laatuun liittyvät aihealueet olivat tuttu kenttä. Tekoäly ei ollut ennestään tekijälle kovinkaan tuttu, mutta tullut esille opintojen aikana eri kursseilla. Ajankohtaisuutensa ja houkuttelevuutensa vuoksi tutkimusaihe valikoitui ja teoria-aineistoon perehtyminen aloitettiin syksyllä 2021.

Tekijä osallistui Mimmitkoodaa-virtuaalitapahtumaan lokakuussa 2021, jossa oli paikalla monia IT-alan organisaatioita. tarkoituksena oli kartoittaa paikalla olevilta yrityksiltä, hyödyntävätkö ne tekoälyä ohjelmistotestauksessa ja onko aiheesta mahdollista haastatella testausammattilaisia. Kyselyllä oli tarkoitus ennen virallista haastattelupyyntöä kartoittaa, minkä verran organisaatioissa on tekoälytestaukseen tietämystä ja kiinnostusta. Tekijä verkostoitui kolmen organisaation edustajan kanssa LinkedIn-verkostossa ja sovittiin, että haastattelut voidaan mahdollisesti pitää keväämmällä, kun tekijän haastattelukysymykset ja aiheen rajaus selkenee.

Kevät asetettiin jatkokeskustelun ajankohdaksi myös tekijän lokakuulle 2021 ajoittuvan lapsen syntymän vuoksi. Tutkimuksen teemaa haluttiin nostaa esille somen avulla sen ajankohtaisuuden vuoksi. Haastateltavien keruun tueksi laadittiin LinkedIn-postaus. Postauksessa korostettiin tutkimuksen aiheen ajankohtaisuutta sekä pyydettiin ottamaan yhteyttä tutkijaan, mikäli verkostossa löytyisi aiheen parissa työskennelleitä ammattilaisia.

Helmi-maaliskuun 2022 aikana tekijä kartoitti loput haastateltavat tutkimusta varten. Kartoitus tehtiin hyödyntämällä LinkedIniä ja osa haastateltavista saatiin hyödyntäen myös lumipallotekniikkaa. Haastateltaviin otettiin yhteyttä

sähköpostitse. Sähköpostissa esiteltiin tutkimuksen aihe, kysyttiin, hyödyntääkö kyseinen organisaatio tekoälyä ohjelmistotestauksessa ja pyyntö osallistua tutkimukseen. Tässä vaiheessa pyrittiin selvittämään, minkälaisiin laadunvarmistuksen ja ohjelmistotestauksen osa-alueisiin organisaatiot tekoälyä hyödynsivät, sillä tämä auttoi tutkijaa rajaamaan myös tutkimuksen teoria-aineistoa. Testaus ja tekoäly itsessään ovat laajoja tutkimusalueita, jolloin vaarana oli, että tutkimuksesta tulisi liian laaja. Osa organisaatioista oli jo hyödyntänyt tekoälyä ohjelmistotestaukseen, kun taas osalla aihe oli vasta suunnitteluasteella.

Jos haastateltavien joukko on liian pieni, aineistosta ei voida tehdä yleistyksiä ja havaita ryhmien välisiä eroja. Mikäli joukko on liian suuri, aineistosta ei voida tehdä syvällisiä tulkintoja. Muutamaa henkilöä haastattelemalla voidaan saada tutkimukselle merkittävää tietoa. Tapaustutkimuksessa jokaiseen tapaukseen kohdistuva tiedonkeruu voi sisältää ison joukon havaintoja, jolloin aineistosta tulee kvalitatiivisesti runsas. (Hirsjärvi & Hurme, 2008, s. 58–59.) Tutkimukseen valikoitui seitsemän organisaatiota ja kahdeksan haastateltavaa.

On tärkeää valmistella haastattelutilanne etukäteen niin, että haastateltava ja haastattelija ymmärtävät, mistä haastattelussa on kysymys (Myers & Newman, 2007). Kaikki haastattelut toteutettiin huhti-toukokuun 2022 aikana. Haastatteluun suostuneille lähetettiin haastattelukysymykset, haastattelun teemarunko sekä linkki haastatteluun noin viikkoa etukäteen, jotta haastateltavat pystyivät perehtymään teemoihin ja selvittämään kysymyksiin vastaukset. Haastattelut toteutettiin etänä Microsoft Teamsin välityksellä johtuen haastateltavien ja tekijän välimatkasta. Jokaiseen haastatteluun varattiin aikaa tunti ja haastattelut nauhoitettiin käyttäen Microsoft Power Pointin screen record -toimintoa, sekä tekijän oman puhelimen äänentallenninta.

5.4 Aineiston analyysi

Analyysi ei ole kvalitatiivisen tutkimuksen viimeinen vaihe, vaan sen tulee olla mukana koko tutkimusprosessin ajan. Analyysi itsessään ohjaa tutkimusprosessia ja tiedonkeruuta. (Kananen, 2008, s. 24.) Monesti tutkimuksen analyysivaihe koetaan tuskalliseksi ja aikaa vieväksi. Tutkija voi viettää jopa kuukausia yrittäessään luoda järjestystä ilmiöihin, etsiessään selityksiä ja yrittäessään tulkita haastateltavien vastauksia oikein. Aineistonkäsittelyyn olisi tärkeää ryhtyä mahdollisimman pian tiedonkeruun jälkeen. Tällöin aineisto on vielä tuore ja inspiroi tutkijaa. Jos käy niin, että tietoja täytyy vielä täydentää, tämä onnistuu helpommin heti haastattelun jälkeen. (Hirsjärvi & Hurme, 2008, s.135.)

Tutkimuksessa käytetään analyysimenetelmänä temaattista analyysia. Temaattisessa analyysissa pyritään tunnistamaan, analysoimaan ja kuvailemaan kvalitatiivisen aineiston teemoja. Cruzes ja Dyba (2011) mukaan temaattinen synteesi on temaattisen analyysin periaate, joka tunnistaa monissa tutkimuksissa toistuvia teemoja ja kysymyksiä. Sen tarkoituksena on tulkita, selittää ja muodostaa johtopäätöksiä teemoista. Tutkimussynteesi on kokoava termi menetelmille, joita käytetään tiivistämään, yhdistämään ja vertailemaan tutkimuksen spesifiä

aihetta tai tutkimuskysymystä. Näiden synteesien avulla voidaan tunnistaa uusien tutkimusten keskeisiä alueita ja tutkimuskysymyksiä, joita ei ole vielä käsitelty tarpeeksi aikaisemmissa empiirisissä tutkimuksissa. (Cruzes & Dyba, 2011.) Temaattinen analyysi sopii tutkimuksen analyysimenetelmäksi, koska tarkoituksena on muodostaa johtopäätöksiä tekoälystä ohjelmistotestauksessa Suomessa.

Aineiston luokittelu on olennainen osa tutkimuksen analyysia. Se luo tutkimuksen kehyksen, jonka varassa haastatteluaineistoa voidaan myöhemmin tulkita, yksinkertaistaa sekä tiivistää. Se on välttämätöntä, jos halutaan vertailla aineiston eri osia toisiinsa tai tyypitellä tapauksia. Luokittelussa jäsennetään tutkittavaa ilmiötä vertailemalla aineiston eri osia keskenään. Tarkoitus on ymmärtää ilmiötä monipuolisesti ja kehittää sellainen näkökulma tai malli, johon luokiteltu aineisto voidaan sijoittaa. (Hirsjärvi & Hurme, 2008, s. 147–148.)

Cruzesin ja Dyban (2011) mukaan temaattisessa analyysissa on kolme luokittelumenetelmää: deduktiivinen, induktiivinen sekä integroitu lähestymistapa. Deduktiivisessa lähestymistavassa luokitteluja asetetaan jo ennakkoon aikaisemmista tutkimuksista ja näitä teemoja pyritään löytämään tutkimuksen aineistosta. Induktiivisessa lähestymistavassa varotaan luokittelemasta aineistoa ennalta. Aineisto käydään läpi tarkasti ja jos käsite tai johtopäätös on ilmeinen, luokitellaan aineisto. Integroitu lähestymistapa on sekoitus deduktiivista ja induktiivista menetelmää. Siinä luokille luodaan yleinen malli, joka ei ole sisältökohtainen, mutta viittaa yleisiin alueisiin. Tutkimuksen aineiston luokittelussa käytetään integroitua lähestymistapaa.

6 TUTKIMUKSEN TULOKSET

Tässä luvussa kuvataan tutkimuksen tulokset. Tulokset pohjautuvat empiirisellä teemahaastattelulla kerättyyn aineistoon. Tulokset esitetään samassa järjestyksessä, kuin tutkimuksen teemat ja kysymykset (Liite 1). Yhteensä kahdeksan ihmistä seitsemästä eri organisaatiosta haastateltiin tutkimukseen. Tutkimukseen haluttiin löytää henkilöitä, joilla on monipuolista taustaa ohjelmistotestauksesta ja kokemusta myös tekoälystä. Haastateltavien edustamat organisaatiot olivat monipuolisia, IT-alan toimijoita ja niiden koot vaihtelivat start-upista globaaleihin suuryrityksiin.

6.1 Haastateltavien taustat

Tutkimuksen teemat ovat haastateltavien taustat ohjelmistojen laadunvarmistuksesta, tekoäly, ohjelmistotestaus sekä tekoäly ohjelmistotestauksessa. Teemat toimivat haastattelukysymysten runkona. Haastattelukysymykset muodostettiin tutkimuksen teemoista tarkentavine kysymyksineen. Kysymykset muotoiltiin avoimiksi, jotta haastateltava kertoisi mahdollisimman monipuolisesti kyseisestä teemasta omin sanoin. Haastateltavien vastauksista on poimittu joitakin sitaatteja, jotta saadaan havainnollistavampi kuva haastateltavien vastauksista ja niiden taustalla olevista konteksteista.

Ensimmäinen teema oli selvittää haastateltavien rooli organisaatiossa, jossa he työskentelevät ja montako vuotta he ovat työskennelleet ohjelmistotestausalalla. Teemalla haluttiin kartoittaa haastateltavan kokemuksesta ohjelmistotestauksessa ja millä tavalla haastateltava voi vaikuttaa tekoälyn hyödyntämiseen ohjelmistotestauksessa. Haastateltavan kokemuksesta pyrittiin saamaan pohdintaa aikaiseksi koskien ohjelmistotestauksen kehitystä tulevaisuudessa.

Haastateltavien tiedot on kerätty Taulukkoon 1 haastattelujärjestyksessä. Haastateltavat on anonymisoitu niin, ettei yksittäisen vastausten perusteella voida tunnistaa haastateltavaa ja hänen edustamaansa organisaatiota.

Organisaation koko on kuvattu liikevaihtoluokassa, jotta saadaan käsitys siitä, kuinka suurissa organisaatioissa tekoälyä testauksessa hyödynnetään.

TAULUKKO 1 Haastateltavien taustatiedot

Haastateltava	**Rooli yrityksessä**	**Kokemus vuosina**	**Organisaation liikevaihtoluokka (€)**
H1	Testausasiantuntija	25	100–500 milj.
H2	Testauksen ja tekoälyn koulututtaja	25	1–5 milj.
H3	Ohjelmistokehittäjä testauksessa	21	100-500 milj.
H4	Testiautomaatiokonsultti	17	50–100 milj.
H5	Datatieteilijä	3 (vuodet tämänhetkisessä organisaatiossa)	50–100 milj.
H6	Testiautomaatiokonsultti	8	50–100 milj.
H7	Yrittäjä, laadunvarmistusvalmentaja	21	100–500 000
H8	Ohjelmistokehittäjä	4	5–10 milj.

Lähes kaikki haastateltavat ovat työskennelleet useamman vuoden jollain tavalla ohjelmiston laadunvarmistuksen, testauksen, testiautomaation tai ohjelmistokehityksen parissa. Haastateltavat H4 ja H5 työskentelivät haastattelun aikaan samassa organisaatiossa ja osallistuivat haastatteluun yhdessä. Tämä tehtiin siksi, koska haastateltavat olivat tekemässä yhteistä projektia liittyen tekoälyyn ja testaukseen. Haastateltava H4:llä oli asiantuntijuutta testiautomaatiosta ja haastateltava H5:llä oli työskennellyt enemmän tekoälytutkimuksen ja datatieteen parissa. Näin ollen nähtiin, että haastatteluun saataisiin rikkaampi kokonaiskuva tutkimusaiheesta, jos haastateltavat osallistuvat haastatteluun yhdessä.

> H3: Mä ylläpidän testiautomaatiota erilaisilla tavoilla. Siihen kuuluu, että mä korjaan testejä tai kirjotan uusia automaattisia testejä tai sit mä ylläpidän osaa siitä testiautomaatoinfrastruktuurista. Mä ehkä kehittelen joitain uusia tapoja testata tai visualisoida joitain ongelmia tai semmosia uusia tapoja tehdä asioita paremmin, ehkä se tällein menee.

> H6: Mä oikeestaan eksyin ensimmäisestä kesätöistä alkaen ohjelmistotestaushommiin ja olen tykännyt näistä niin en ole hakeutunut muunlaisiin tehtäviin, mut mä kutsun itseäni vähän semmoseks työkalusepäksi. Mä teen työkaluja niille, jotka tekee testaamista tai ylipäätänsä kaikkeen, devopsiin tai automaatioon liittyen.

> H1: Mä olen 25 vuotta ollut testausalalla. Et mä oon vaihtelevasti tehnyt erilaisia rooleja tässä vuosien varrella. Mä olen ollut tutkijana välillä, sit mä oon ollut esimiesroolissa välillä ja nyt mä oon taas asiantuntijaroolissa.

Suurin osa haastateltavista tekevät konkreettista testausta ja testiautomaatiota asiakasprojekteissa, mutta haastateltavien työnkuvassa oli myös testiautomaation ylläpitotehtäviä, testaukseen liittyvää kouluttamista, tekoäly- ja datatiedekonsultointia ja ohjelmistokehitystä. Haastateltavien organisaatioiden ydinliiketoiminta vaihteli myös jonkin verran, vaikka kaikki organisaatiot tekivät ohjelmistokehitystä ja testiautomaatiota. Painotuksia oli esimerkiksi sulautetuissa järjestelmissä, tietoturvaratkaisuissa, testiautomaatiokonsultoinnissa ja ohjelmistojen laadunvarmistuksessa. Täten saatiin kattava otanta erityyppisistä organisaatioista ja tutkimustapauksista.

6.2 Tekoäly

Haastateltavilta kysyttiin, miten he määrittelisivät tekoälyn ja mitä tekoäly heidän mielestään on. Tähän haluttiin saada haastateltavien näkemyksiä, koska tutkimuksen kirjallisuuskatsauksessa tekoälyn määritelmät vaihtelivat paljon. Haastateltavien vastaukset vaihtelivat paljon ja monelle tekoälyn määrittely omin sanoin tuntui haastavalta. Haastateltavat H3 ja H8 mainitsivatkin, että kävivät tai halusivat käydä tarkastamassa Googlesta, miten tekoäly määritellään oikeasti. Osa haastateltavista rinnastivat tekoälyn määritelmää osana heidän työtään, eli miten tekoäly näkyy heidän työssään. Esimerkiksi haastateltava H7 määritteli tekoälyn ohjelmistotestauksen kontekstissa.

> H7: Mä määrittelisin tekoälyn niinku jos se olisi oikeasti tekoälyä niin sellaiseksi, että se osaisi tarjota ihmiselle vaihtoehtoja. Ja osaisi myös testauskontekstissa oikeasti löytää uutta tietoa siitä tuotteesta, ettei vaan perinteinen testiautomaatiolähinnä tarkistaa asioita, että onko jotkut outputit tietyissä rajoissa tai virheen käsittely oikein, mutta sitten tekoäly olisi sitten enemmän sellainen tietoa tuottava kumppani.

Haastateltava H5 totesi, että tekoälyn älykkyyttä voi olla erilaista. Esimerkkeinä hän käytti, että miten koneen voi tunnistaa asioita kuvainformaation perusteella ja voiko fyysinen robotti liikkua jossain ympäristössä havainnoiden ympäröivää maailmaa autonomisten autojen tavoin. Vastauksessa oli samankaltaisuutta tekoälyteorian kanssa, sillä haastateltava H5 mainitsi tekoälyn alalajeja, kuten robotiikka, NLP, konenäkö ja koneoppiminen. Haastateltavan toimenkuva painottuikin tekoälytutkimukseen, jolloin vastauksessakin heijastui tekoälytieteelle ominaisia seikkoja.

Haastateltava H8 käytti vastauksessaan esimerkkinä autoa, jonka tuulilasiin on laitettu laite, joka kuvaa ajoreitin teitä keräten tietoa teiden kunnosta tallentaen teiden sijainnin. Tämän jälkeen kerätty aineisto ladataan pilvipalveluun, jossa tekoäly tunnistaa, minkälaiset kohdat ovat huonoja asfaltissa ja sen perusteella kartoittaa, missä paikoissa teitä täytyy korjata. Tekoäly tässä tapauksessa siis oppii, millainen on hyvä tie ja millainen on huono tie. Haastateltava H8 työskentelee sulautettujen järjestelmien parissa, jolloin tekoälyn näkökulmakin oli rinnastettavissa fyysisten laitteiden testaukseen.

> H8: Se siis on kykyä käsitellä on iso määrä dataa ja löytää just se, mitä me tarvitaan.

Puolet haastateltavista nostivat esille datan analysoinnin ja päätöksenteon datan pohjalta. Näihin piirteisiin rinnastettiin vahvasti termi älykkyys. Tekoäly kykenee käsittelemään suurta määrää dataa ja tarjota ihmiselle vastauksia tai vaihtoehtoja, jotka eivät ole olleet etukäteen tiedossa. Haastateltava H4 kertoi, että hän ymmärtää tekoälyn ei ihmisen tekemäksi älykkäiksi valinnoiksi ja jos sen voi ymmärtää älykkääksi tai käyttömukavaksi, niin se riittää hänelle tekoälyyn. Haastateltavat H2 ja H6 kiteyttivät päätöksenteon ja ongelmanratkaisun siten, että tekoäly etsii itse ratkaisua ongelmaan annettujen tietojen perusteella. Sen sijaan, että ratkaisu on koodattu ongelmaan valmiiksi, tekoäly oppii ratkaisemaan itsenäisesti ongelmia ja tulee prosessissa samalla paremmaksi ja älykkäämmäksi.

> H3: Ehkä se on jotain semmosta asiaa, jota on kenties hankala hahmotella pelkästään koodilla tai tämmösillä säännöstöillä, vaan tarvii jonkinnäköistä erilaista ratkaisumallia ratkaistakseen jonkun ongelman. Ja johon voi niinkun käyttää dataa, olemassa olevaa dataa sen ongelman ratkaisuun. Se on tavallaan niinku se, ehkä se jotain semmoista on.

Kaksi haastateltavaa rinnastivat tekoälyn ihmiselle ominaisiin toimintoihin. Haastateltava H2 mainitsi, että jonain päivänä tekoäly voi olla yhtä älykäs olento, kuin ihminen. Tekoälyn kehityskulku voi mennä tähän suuntaan, mutta toivon mukaan sillä tavalla, että tekoälyn kehitykselle asetetaan rajoja, ettei synny tilanne, jolloin tekoälyä ei pysty enää hallitsemaan.

> Ihmisten toiminnan matkimista tietokoneiden toimesta. – – Niinku näitä tyypillisiä kognitiivisia prosesseja, joita ihminen soveltaa. Päätöksenteko. -Haastateltava H1

Temaattista analyysia hyödyntäen, vastausten perusteella muodostetaan ensimmäinen empiirinen johtopäätös eli EC (*eng. Empirical conclusion*) siitä, mitä tekoäly haastateltavien mukaan on.

> EC1: Tekoälyn älykkyys on ongelman ratkaisua ja päätöksentekoa perustuen annettuun dataan.

Viisi kahdeksasta haastateltavasta rinnastivat tekoälyn koneoppimiseen, eli tämän päivän tekoäly lähes sama asia, kuin koneoppiminen. Haastateltavat H5 ja H2 kuitenkin tarkensivat, että koneoppiminen on yleisin tapa toteuttaa tekoälyratkaisuja, eli koneoppiminen on väline tai työkalu älykkäiden ratkaisujen toteutukseen.

> H2: Ei oo sinänsä mielipidekysymys vaan tekoälyn voi määritellä monella tavalla ja yks on ainakin se, että koneoppiminen tänä päivänä on yks sen synonyymi, mutta se on vaan yksi tapa tehdä tekoälyä, se yleisin tapa et annetaan algoritmin tavallaan itse oppia ja sit se pystyy sen jälkeen suoriutuu ja sit se suoriutuu tehtävistä mistä se ei oo suoriutunut ennen sitä ja jatkuvasti oppii lisää, jolloin siihen tulee se älykkyys. Et silleen tekoäly näin ollen on sama asia, kun koneoppiminen.

> H6: Ehkä yleisesti just tää että et aina kun kone ratkaisee sen ongelman silleen että se ohjelmoija ite ei tavallaan ole tiennyt sitä ratkaisua tavallaan se ratkaisu siihen ongelmaan ei oo ollu etukäteen tiedossa, eli sinne on vaan annettu ohjeet sille koneelle keksiä, löytää se ratkasu siitä tavallaan kentästä. Eli kaikki sen tyyppiset ongelmat on musta tekoälyä mutta sitten käytännössä kun puhutaan asioista kun yleensä puhutaan tekoälystä niin sit käytännössä puhutaan koneoppimisesta.

> EC2: Tekoälyn älykkäät toiminnot perustuvat koneoppimiseen.

Nämä löydökset muodostavat toisen empiirisen johtopäätöksen sekä pääasiallisen empiirisen johtopäätöksen (*eng. Primary empirical conclusion*) siitä, mitä tekoäly on.

> PEC1: Tekoäly voi ratkaista ongelmia ja tehdä päätöksiä annetun datan perusteella. Tekoälyn toiminnot perustuvat koneoppimiseen.

6.3 Ohjelmistotestaus

Tutkimuksen toinen pääteema ohjelmistotestaus jaettiin kolmeen tarkentavaan kysymykseen. Haastateltavia pyydettiin kuvailemaan omin sanoin jonkun ohjelmistoprojektin testausprosessia sekä kuvailemaan, mitkä osat testauksessa ovat manuaalisia ja mitkä automatisoituja. Haastateltavien vastauksista kävi ilmi, että kysymys oli vaikeasti rajattava, joten jotkut haastateltavat kokivat kysymykseen vastaamisen haasteellisena. Haastateltavat kertoivat testauksesta yleisellä tasolla ja millaisena testaus näyttäytyy heidän työssään. Osa haastateltavista kertoi testauksesta korkeammalla tasolla, erityisesti laatutietoajattelun näkökulmasta. Jotkut haastateltavat kertoivat testauksesta hyvin yksityiskohtaisesti, miten ohjelmistotuotteen testit ajetaan.

Useimmat haastateltavat kokivat sanan testausprosessi termiksi, joka irrottaa ohjelmistotestauksen erilliseksi komponentiksi ohjelmistokehityksestä, mitä se ei haastateltavien mukaan ole. Kaksi haastateltavaa kiteytti, että testausprosessi ei välttämättä ole enää sanana kuvaileva, koska testaustiimit ja ohjelmistokehitystiimit eivät enää toimi erikseen. Testaus kuuluu olennaisena osana ohjelmistokehitysprosessia.

> H3: Mun mielestä jos kuvaillaan testausprosessia, tai testausprosessi ei ole oikeestaan sanana kuvaileva, kannattaa mieluummin kuvailla sitä ohjelmistokehitysprosessia koska se testaus kuuluu olennaisena osana sitä ohjelmistokehitysprosessia.

> H7: Tää on kanssa muuttunut aika paljon tossa vuosien varrella, että. Että siinä, missä niinku aikaisemmin oli testaustiimejä ja softakehitystiimejä, niin nää on niinkun aika paljon sulautunut yhteen, että me harvemmin ollaan mukana enää sellaisessa hankkeessa, missä nää toiminnot olisi erikseen.

> H1: Mä en oo ikinä välittänyt siitä itse prosessista viimekädessä. - - Eli tavallaan mulle testausprosessi ei ole sellainen asia, joka tulee jostain aina “näin se aina menee”,

> suunnitellaan, toteutetaan, tehdään. Vaan se on enemmän sellainen aktiivinen ajattelun tulos siitä, että me tarvitaan jotain tietoa sen päätöksenteon tueksi, kun me rakennetaan softaa ja testaus on se työkalu, joka tuottaa tätä tietoa.
>
> EC3: Testaus ja ohjelmistokehitys eivät ole erillisiä toimintoja, vaan testaus on osa ohjelmistokehitystä.

Tekijä tarkensi joidenkin haastateltavien kohdalla kysymystä siten, että jos on olemassa oleva tai täysin uusi ohjelmistokehitysprojekti, millä tavalla tähän projektiin laatua lähdetään rakentamaan. Vastausten perusteella testausta ja laatua tulisi lähteä rakentamaan ohjelmistotuotteelle alusta asti ja jatkaa sitä koko ohjelmiston elinkaaren ajan. Kaksi haastateltavaa mainitsivat, että vaikka testaus tulisi ottaa mukaan organisaatioon heti alusta saakka, näin ei kuitenkaan aina ole. Organisaatio saattaa myöhemmin oppia kantapään kautta, että testaus olisi täytynyt ottaa mukaan jo alusta saakka.

Haastateltava H1 kertoi, että on päätynyt organisaatioihin, joissa on tehty tiettyä ohjelmistotuotetta jopa vuosikymmeniä ja testauksen kautta lähdetään ratkaisemaan ohjelmiston ongelmia. Tämä saattaa tapahtua vasta siinä vaiheessa, kun esimerkiksi asiakas törmää virheilmoituksiin ohjelmistotuotetta käyttäessään. Haastateltava H2 totesi, että ihannetilanteessa testaus ja laadunvalvonta otetaan mukaan ohjelmistokehitykseen mahdollisimman aikaisessa vaiheessa, mielellään jo vaatimusten määrittelyvaiheessa. Testausprosessin korostettiin vaihtelevan asiakkaan ja asiakkaan tuotekehitysprosessien mukaan. Eroja on myös siinä, onko kyseessä käyttöönottoprojektista vai tuoteprojektista. Kolme haastateltavaa kertoivat, että yksinkertaisimmillaan testausprosessi on yhdistelmä automatisoituja yksikkö- ja integraatiotestejä, testiautomaatiota sekä tutkivaa testausta, joka päättyy jonkinlaiseen hyväksymistestausvaiheeseen.

Viisi haastateltavaa totesivat testauksen olevan vahva osa ketterää ohjelmistokehitystä vaatimusten määrittelystä alkaen ja testit tulisi kytkeä osaksi jatkuvaa julkaisua. Testaus on kytköksissä ohjelmistoon, jolloin tulisi ymmärtää, mitä ollaan tekemässä, mihin tarkoitukseen ja kenelle. Osa haastateltavista kertoi, miten eri testaustavat sisältyvät ohjelmistokehitykseen. Haastateltavien mukaan testausta tapahtuu jo ohjelmistokehitysvaiheessa ohjelmistokehittäjien toimesta yksikkötesteillä ja mahdollisesti integraatiotesteillä.

Haastateltavat mainitsivat testaustyypeistä tutkiva testaus, fyysisten laitteiden testaus, end-to-end testaus, suorituskykytestaus, testiautomaatio, performanssitestaus, yksikkötestaus, integraatiotestaus, regressiotestaus ja testiautomaatioympäristön ylläpito. Haastateltava H4 kertoi, että testaajan rooli on muistuttaa asioiden olemassaolosta ja luodaan pohjaa, joka mahdollistaa testauksen. Haastateltava H1 totesi, että testauksen avulla rakennetaan aikaisemman päälle parempaa ja laatutietoa saadaan päätöksenteon tueksi.

> H1: Et mulle testausprosessi on sellaista, että piirrän sen sellaisena laatikkona, et sulla on inputti ja outputti. Inputti on hyviä ihmisiä ja jotain materiaalia, josta ne voi päätellä ja oppia asioita, josta ne ei tiedä vielä, ja outputti on parempia ihmisiä, ja sit jotain materiaaleja, jotka auttaa meitä tulevaisuudessa tekemään asioita paremmin. Se on, miten mä ajattelen testausta, et se on vaan laatikko, joka tuottaa tietoa.

Jatkuvasta toimituksesta päästiin tarkentaviin kysymyksiin, jotka olivat mitkä toiminnot testauksessa tehdään edelleen manuaalisesti ja mitkä toiminnot ovat automatisoitu. Usea haastateltava tarkensi, etteivät manuaalinen ja automatisoitu testaus ole täysin erikseen jaoteltavia asioita. Haastateltava H1 luokitteli nämä asiat englanniksi attended ja unattended -toiminnoiksi. Attended on toimintaa, joka vaatii ihmisen läsnäoloa ja unattended on toimintaa, joka voidaan tehdä ilman ihmisen valvontaa. Haastateltavan H1:n mielestä testauskentällä ja projekteissa on vanhanaikaista ajattelutapaa siitä, että automaatio on sitä, mitä ajetaan sillä aikaa, kun kehittäjät eivät ole läsnä. Automaatio ja tekoäly ovat enemmän niin sanottua attended-toimintaa, joka on tekemisen tukena ja tehostamisena keventäen työkuormaa tai vähentämässä virheiden määrää.

Haastateltava H2 mukaan jo käyttäjätarinamäärittelyvaiheessa tulisi miettiä jo esimerkkitestejä ja testiautomaatiota tulisi lähteä rakentamaan esimerkkitesteihin pohjaten. Haastateltavista neljä korostivat, testiautomaatio on rinnakkaista toimintaa manuaalisen testauksen kanssa. Kehittäjien tekemät yksikkötestit ja automaattitestit tulisi kytkeä osaksi jatkuvaa julkaisua, jolloin myös ohjelmistotuotteen koodi laajenee vaihe vaiheelta. Haastateltava H3 totesi, että testiautomaatio toimii ikään kuin turvaverkkona varmistaen, että kaikki toimii muutoksista huolimatta. Myös haastateltava H4 teki saman johtopäätöksen.

> H2: Ja sitten testauksen kannalta se, et kaikki testit, mitä siihen tehdään niin ne automatisoidaan ja niitä pyöritetään ihan koko ajan. Että koko ajan varmistetaan, että mikään ei oo hajonnut. Eli tehdään jatkuvaa regressiotestausta.

Haastateltava H6 korosti myös regressiotestauksen automatisointia.

> H6: No siis ihan yleisesti se mitä, ja varsinkin tämmösessä konsulttikeisseissä usein tehdään, on se, että meillä on vanhakantasta. Varsinkin kun puhutaan, et, kun mä kirjaudun tästä, niin siinä on aika paljon regressiotestaamista ja tavallaan se regressiotestaamisen automatisointia, et siellä on vanhaa manuaalista, mikä tavallaan automatisoidaan missä siellä on yleensä eniten tekemistä. – Haastateltava H6

> H4: Tässä tulee hyvin vahvasti se jako manuaalinen automaatio on tuossa ketterässä prosessissa, että silloin, kun ollaan sitä jotain toiminnallista tikettejä tekemässä ensimmäistä kertaa, se saadaan sprintissä niinku valmiiksi. Siihen yleensä kuuluu sitten manuaalinen testaaminen, että se, mikä on kehitetty sprintin aikana, toimii. Tota testataan manuaalisesti toimivaksi sprintin aikana, että se voidaan toimivana julkaista, mutta siinä rinnalla kulkee koko ajan tää automatisointiprosessi, koska se manuaalinen testaaminen sehän testaa sen vaan kerran. Mutta sehän ei takaa sitä että se toimii taas 2 viikon päästä, niin siinä on hyvin tärkeätä saada sitten se automaatioprosessi siihen viereen.

> EC4: Testaus tulisi sisällyttää ohjelmistokehitykseen alusta alkaen.

> EC5: Testiautomaation avulla varmistetaan, että ohjelmiston ominaisuudet eivät hajoa kehityksen aikana.

Moni järjestelmä on julkaistu johonkin pilvipalvelualustaan, kuten Microsoft Azureen tai AWS:ään. Pilvialusta päivittyy taustalla jatkuvasti, jolloin automaattitestien avulla varmistetaan, toimiiko ohjelmisto edelleen päivityksistä huolimatta. Tällainen testaus on jatkuvaa regressiotestausta, johon myös haastateltava H2 viittasi. Kolme haastateltavaa määrittelivät manuaalisen testauksen tutkivaksi testaukseksi, jossa ihminen pyrkii löytämään ohjelmistosta ominaisuuksia, joita testiautomaatio ei löydä.

Haastateltava H3:n mukaan hyvä testaaja osaa lukea ohjelmistotuotetta oikein yllättävänkin pienistä palautteista. Hyvä testaaja osaa vaihtaa testauksen kurssia johonkin asiaan. Tuote ikään kuin puhuu testaajalle ja testaaja kykenee kuuntelemaan ja löytämään tämän perusteella tuotteesta mielenkiintoisia asioita. Haastateltava H7:n mukaan testausala on mennyt yhä enemmän teknisemmäksi. Hän totesi myös, että testaajan ajattelutapa eroaa kehittäjän ajattelutavasta siten, että testaaja ei tee testausta pelkästään toiminnallisuuksien kautta vaan myös tutkivan testauksen kautta. Kolme haastateltavaa totesivat testaajan ajattelevan enemmän käyttäjänäkökulmaa, kuin taas ohjelmistokehittäjät teknistä näkökulmaa.

> H3: Niin tavallaan tutkiva testaus, mä en tykkää sanasta manuaalinen testaus, kun en usko et tavallaan, se sana tuntuu jotenkin rumalta. Vaan tutkiva testaus on enemmänkin sillein, se ratkaisee samaa ongelmaa, mutta se ratkaisee sitä vähän niinkuin eri puolilta, se yleensä löytää niinkuin enemmän vikoja, se löytää yleensä parempia vikoja, se löytää mielenkiintoisempia vikoja, se löytää ehkä tämmösiä testikeissejä helpommin ja jos ajattelee sillein, että tutkiva testaus antaa niinkuin syvyyttä siihen testaukseen.

> H7: Mutta tosiaan se roolitus on vähän muuttunut. Että se tapahtuu siellä kehitystiimin sisällä, eikä niinkään niinkun vaiheessa, koska ketterästi yleensä toimitaan niin niin tää on enemmän tai vähemmän jatkuvaa toimitusta ja jatkuvaa kehitystä tää tekeminen.

Haastateltava H7 kertoi myös, että hän näkee kasvavana roolina QA-painotteisen (*eng. Quality Assurance*) testausroolin, jossa henkilö pystyy osallistumaan myös järjestelmän koodaamiseen. Tällaiset henkilöt ovat haastateltava H7:n mukaan olla nimikkeellä Software Engineering In Test tai Software Developer In Test. Hän nosti myös esiin tärkeän seikan:

> H8: Vaikka sulla olisi kuinka paljon polkukattavuutta sun koodissa yksikkötestien tai muiden testien kautta, niin se ei kerro sulle, teetkö sä oikeeta tuotetta. Että siihen tarvitaan niinku konteksti- ja liiketoimintaosaamista.

Haastateltavien mukaan testiautomaation avulla ohjelmistotestaukseen saadaan laajuutta ja skaalaa, mihin tutkiva testaus ei taivu. Testiautomaation avulla voi ajaa esimerkiksi samat testit erilaisissa käyttöjärjestelmissä, jolloin voidaan varmistua, että tuote toimii erilaisissa ympäristöissä. Neljä haastateltavaa totesivat, että testiautomaation avulla ohjelmistoon kehitetyt ominaisuudet pidetään toimivana. Haastateltava H4 vertasi automaattitestejä tuotteen määrittelyihin ja totesi, että tänä päivänä parhaimmillaan automaattitesti ovat niin selkeitä,

että niiden avulla voidaan tulkita, miten järjestelmän kuuluu toimia. Testiautomaatioon käytettävistä työkaluista Robot Framework oli suosituin. Haastateltava H4:n mukaan perinteisessä funktionaalisessa, eli toiminnallisessa testauksessa 90 % markkinoista käyttää testiautomaatiossa Robot Frameworkia. Jenkins on edelleen jatkuvan julkaisun työkaluna suosittu, mutta myös Github Actions mainittiin nousevana automaattisen julkaisun työkaluna.

> EC6: Testiautomaatio ja manuaalinen testaus ovat rinnakkaisia, eivätkä toisistaan erillisiä toimintoja.

> EC7: Tutkivalla testauksella pyritään löytämään ohjelmistosta ne viat, joita testiautomaation avulla ei löydetä.

Kaksi haastateltavista työskentelivät pääosin sulautettujen järjestelmien parissa. Sulautetussa järjestelmäkehityksessä tehdään järjestelmiä johonkin tiettyyn laitteeseen tai laitteisiin. Esimerkiksi älypuhelin tai GPS-paikannin ovat sulautettuja järjestelmiä.

> H6: Tossa tiettyä sellasta niinkun tavallaan laitteen tai infraa, sellasta niin siellä on vielä semmonen laadunvarmistuksen taso, et se on niin kriittistä, se on niin kallista korjata sitä. Tavallaan fyysisiä laitteita, se on tosi tärkeetä mut tosta tuli mieleen testaamisesta, että on pääasiassa manuaalista prosessia, siellä on ihan niitä laitteiden fyysistä testaamista, et siellä tapahtuu myös sitä, että siellä yritetään katsoa, että millasia olosuhteita ne laitteet kestää.

Kahden haastateltavan mukaan sulautettujen järjestelmien testaus sisältää fyysisten laitteiden ja niiden konfiguraatioiden testausta. Yksi haastateltavista kertoi esimerkin, että sulautettujen järjestelmien kehityksessä automatisointia on sovellettu manuaalitestauksessa siten, että excel-kirjauksia laitteiden fyysisistä testauksista on pyritty korvaamaan automatisoimalla tulosten analysointi, raportointi, vertailu ja lokien keruu.

Haastateltava H8 kertoi, että testiautomaatiota hyödynnetään hänen edustamassaan organisaatiossa raskaan kaluston konfiguraatioiden testauksessa. Testauksella varmistetaan, että sulautettu järjestelmä toimii kaikissa konfiguraatioissa ja toimintojen olemassaolo ei vaikuta järjestelmän suorituskykyyn. Tähän käytettiin haastateltavan H8:n mukaan HIL-testiautomaatiotyökalua (*eng. Hardware in the Loop*). HIL:in avulla testataan raskaan kaluston sulautettuja järjestelmiä ja niiden toiminnallisuutta. Testauksessa hyödynnettiin Jenkins-automaatiopalvelinta, joka käynnistää testausprosessit aikataulun mukaan ja hakee myös testattavat konfiguraatiot versionhallinnasta. Jenkins tekee myös automaatiotestien filtteroinnin, eli valitaan vain ne testit, jotka pystytään ajamaan kyseisellä konfiguraatiolla.

> H8: Jos traktorissa on real drive station ja se on kun on sellaisia traktoreita, että kuljettaja voi kääntää penkin taaksepäin ja siellä on takana myös kaasu ja ohjauspyörä ja pystyy ajaa taaksepäin. Ja jos me tunnistetaan, että konfiguraatiossa on sellainen real drive station, tässä tapauksessa me testataan, että jarrut jotka ovat takana, ne myös toimivat ja näin edelleen.

Haastateltava H8 mukaan manuaaliset vaiheet ovat testattavien konfiguraatioiden määrittely, eli mitkä konfiguraatiot halutaan testata, milloin testit käynnistetään ja minkä verran testejä halutaan ajaa. Esimerkiksi jos halutaan ajaa pelkästään turvallisuuskriittiset testit, testiajo tapahtuu tällöin nopeammin, kuin jos ajettaisiin kaikki olemassa olevat testit.

> PEC2: Ohjelmistotestaus on osa ohjelmistokehitystä. Testiautomaatio toimii ohjelmiston laadun turvaverkkona. Tutkivalla testauksella pyritään löytämään ne ominaisuudet, joita testiautomaation avulla ei löydetä.

6.4 Tekoäly testauksessa

Tutkimuksen kolmannessa teemassa tarkastellaan, millä tavoin tekoälyä hyödynnetään ohjelmistojen testauksessa. Osa haastateltavien organisaatioista ei vielä ollut ottanut tekoälypohjaisia ratkaisuja osaksi ohjelmistotestausta, mutta olivat vähintäänkin tehneet jonkinlaisen kirjallisuuskatsauksen tai selvityksen aiheeseen liittyen. Tästä syystä kysymykset muotoiltiin sen mukaan, olivatko tutkimustapaukset jo hyödyntäneet tekoälyä testauksessa vai vielä suunnitteluvaiheessa

6.4.1 Nykyiset haasteet testauksessa

Teema jaoteltiin noin kuuteen tarkentavaan kysymykseen, joista keskusteltiin haastateltavien kanssa vapaassa järjestyksessä. Ensimmäinen kysymys oli, minkälaisia haasteita testauksessa on, joihin toivotaan tekoälyn avulla ratkaisua. Kysymyksellä haluttiin selvittää haastateltavien kokemuksia testauksen tämänhetkisistä haasteista ja voidaanko niitä helpottaa tekoälyn avulla.

Kaksi haastateltavaa (H4 ja H5) olivat tehneet organisaatiossaan selvityksen testauksen tämänhetkisistä haasteista ja tekoälyn hyödyntämismahdollisuuksista. Haastateltava H4 totesi, että tekoälyltä toivottiin ratkaisua vähän kaikkeen ja tekemään asioita hienojakoisemmaksi ja tarkemmaksi. Erityisesti mitä epämääräisempi ja jaetumpi testiympäristö on, niin sitä enemmän siellä pitäisi olla älykkyyttä mukana. Haastateltava H2 suhtautui tekoälyn tuomiin mahdollisuuksiin optimistisesti ja totesi, että lähestulkoon kaikkiin asioihin voi hyödyntää tekoälyä ja tarvittavat työkalut ovat jo käytännössä olemassa.

Keskusteltaessa testauksen haasteista yli puolet haastateltavista nostivat esiin testiajot ja niistä saadun palautteen viiveen. Järjestelmän testiajossa voi olla satoja tuhansia testitapauksia, jotka laitetaan ajoon. Haastateltavien mukaan testiajo saatetaan laittaa toteutukseen illalla ja seuraavana aamunakaan se ei välttämättä ole valmis. Joskus testiajon valmistuminen saattaa mennä viikonlopun yli. Tämän takia palautteen saanti testeistä kestää tarpeettoman kauan ja palaute on tällöin jo myöhässä, kun järjestelmässä ehditään testiajon aikana rikkoa ominaisuuksia.

> H1: Niin palautteen aikaoptimointi, et tavallaan se, että sulla on 10 000 testikeissiä, jotka automaatio ajaa sulle. Kun se käytännössä alkaa illalla 11:00 raksuttaa ja aamu kuudelta se ei oo vielä valmis ja 6:30 ensimmäiset tyypit aloittavat työpäivänsä. Niin se palaute on jo tavallaan myöhässä sen ensimmäisen päivän osalta. Niin yleensäkin se, että me joudutaan odottamaan kokonainen päivä, et me saadaan yöllä niitä tuloksia, niin se on ihan kestämätön tilanne.

Haastateltavien mukaan olisi tärkeä saada tieto siitä, mitkä testit ovat tärkeämpiä testata ensin. Tärkeillä testeillä tarkoitettiin sitä, että mitkä testitapaukset todennäköisimmin hajoavat. Tällöin testiajossa voitaisiin priorisoida tärkeimmät testit ajoon ensin tai saada niistä palaute nopeammin. Loput testit, jotka eivät todennäköisesti hajoa, saisivat olla testiajon suoritettavana pidempään ja niiden palautteen saanti ei olisi aikakriittistä. Puhutaan siis testitapausten priorisoinnista. Haastateltava H2 korosti, että priorisointi ja älykkäämpi automaatio ovat hyödyllisimmät kohdat testauksessa, johon hyödyntää tekoälyä.

> H2: Jos sä ajat regressiotestinä vaikka tuhat testiä joka päivä ja jos niistä hajoo vaikka sata testiä, niin sit sä käytät x-tuntia niiden korjaamiseen. Sun kalliit testiautomaatiokonsultit vaan korjaa skriptejä jonkun tuntimäärän joka ikinen päivä, niin mitä jos ne ei hajoiskaan? Sä säästäisit kaiken sen ajan, sen viis tuntia päivässä korjailua, niin siinä miettii, mikä olis se vaihtoehtoiskustannus, jos vaikka kuitenkin korjattais niitä nykyisiä skriptejä, ettei ne hajois niin se on se business-case, mitä me yritetään kokoajan tehdä.

Haastateltava H6 kuvaili testiajojen ongelmaa sulautettujen järjestelmien kehityksessä:

> H6: Mut sit ku puhutaan asioista kun on fyysisiä laitteita ja koneita niin niiden, ku niitä aletaan testaa siellä loppuvaiheen testeissä niin niissä on se vika että olis hirveen kiva saada palaute nopeemmin. Mutta kun on rajoitteita, et niitä ei voi kopioida niitä koneita, ne fyysiset laitteet on rajoite, et resursseja on vähän niin sitten se, että se oikeesti vie aikaa. Et ku siellä on jotain asioita, mitkä liikkuu, niin sit liikkuvien asioiden liikkuminen niin sitä ei silleen voi nopeuttaa niin se tarkottaa sitä, et sen regressiotestisetin ajaminen menee viikonlopun yli. – Haastateltava H6

> EC8: Testiajojen tuottaman palautteen saanti kestää liian kauan.

> EC9: Testitapausten suorittamisjärjestystä tulisi voida priorisoida.

Viisi haastateltavaa painotti testiautomaation kyvyttömyyttä tehdä älykkäitä ratkaisuja. Tällä tarkoitettiin sitä, että testiautomaatio tekee vain ne asiat, jotka testaaja on käskenyt sen tekemään. Tällöin se ei ota huomioon niitä asioita ohjelmistossa, jotka muuttuivat edelliseen testiajoon verrattuna. Haastateltava H1 korosti myös automaation ylläpitämiseen vaadittavaa aikaa. Tällä hän ei tarkoitettu testiautomaation hajoamista vahingossa, vaan sitä, että maailma muuttuu ympärillä, jolloin myös automaatiotyökalujen tulee mukautua muutoksiin.

> H1: Eli tyypillisesti sehän vie meiltä, mitähän se vei, meillä oli neljä täyspäiväistä testaajaa niin puolet niistä suunnilleen teki pelkästään sitä automaation ylläpitämistä, eli me ei menty tavallaan eteenpäin, vaan pidettiin automaatiota hengissä.

> EC10: Testiautomaatio ei mukaudu ohjelmiston muutoksiin itsenäisesti.

Kaksi haastateltavista mainitsi, että käyttöliittymätestaus on osa-alue, jonka voisi jättää tekoälyn hoidettavaksi. Haastateltava H7 totesi, että käyttöliittymätestaus on yleisesti kallein ja hankalin osa tehdä testausta, koska käyttöliittymätestit on vaikea saada pysymään kunnossa. Tekoälytestaukseen erikoistuneet firmat, kuten Applitools ja Test.ai ovat keskittyneet erityisesti käyttöliittymätestaukseen, minkä useampi haastateltavakin mainitsi.

> EC11: Käyttöliittymätestit hajoavat helposti ja testien ylläpidettävyys on työlästä.

Haastateltava H8 koki vaatimusten määrittelyn asiaksi, johon tekoäly voisi tuoda helpotusta. Hän tarkensi, että tekoäly tarkastaisi vaatimukset ja siitä saadun datan perusteella tarkistaisi, että järjestelmä on rakennettu vaatimusten mukaan. Haastateltava H8 mainitsi, että isoin haaste HIL:in, kanssa on itse HIL:in kehittäminen. Hän kertoi, että monesti testit eivät mene läpi sen takia, että HIL:issä on joku vika, eikä järjestelmässä.

> H8: No tässä tapauksessa voisi sanoa, että tekoäly voisi tutkia HIL:in järjestelmän ja etsiä puutteet ja mahdollisesti tehdä korjaukset maiden tasolla tai raportoida rautatason ongelmista. Ja tässä tapauksessa meillä on tarkka kuva, mitä oikeasti pystymme testata ja mitä ei.

Haastateltava H1 mainitsi testitapausten luonnin manuaalisen kuvauksen perusteella. Tekoäly voisi määritellä tiketin tasolta ihmisen tahtotilan siitä, mitä hän on halunnut järjestelmässä muuttaa, mikä toiminta on ollut tarkoituksellista ja mikä tahatonta. Haastateltava H1 kertoi, millaisia haasteita uransa aikana testauksessa on ilmennyt ja pohti niihin perustuen, millaisissa asioissa tekoäly kykenisi haasteita lieventämään. Erityisesti turvallisuuskriittisillä aloilla standardit saattavat vaatia, että testidokumentaatiossa on kirjattu tarkasti ylös, mitkä asiat on tehty ja testattu.

Haastateltava H1:n mukaan tällainen dokumentaatio ei ole tärkeää tuotteen kehittämisen tai testauksen kannalta, mutta dokumentaatiota on oltava laatustandardien vuoksi. Kyseessä on siis rutiininomainen työ, joka itsessään ei tuota lisäarvoa asiakkaalle tai organisaatiolle, mutta on pakollinen tehtävä. Haastateltava H1 toivoisi, että tekoäly voisi helpottaa tällaisten dokumentaatioiden laadintaa esimerkiksi tekstintunnistuksen avulla.

> EC12: Testidokumentoinnin luonti ja vaatimusten määrittely koettiin työlääksi.

PEC3: Testauksessa koettuja haasteita ovat testiajoista saadun palautteen viive, käyttöliittymätestien ylläpidettävyys sekä testidokumentoinnin tuottaminen manuaalisen kuvauksen perusteella.

6.4.2 Tekoälyn hyödyntämisen testauksen vaiheisiin

Seuraavat kysymykset haastateltaville olivat, minkälaista tekoälyä testauksessa on hyödynnetty tai suunnitellaan hyödynnettävän sekä mihin testauksen vaiheisiin tekoälyä on hyödynnetty. Haastattelutilanteessa testauksen aiheuttamista haasteista siirryttiin luontevasti keskustelemaan tekoälyn tuomista ratkaisuista testaukseen. Puolet haastateltavista kertoivat, että heidän organisaatioissaan on tehty vähintäänkin kokeiluja tekoälyn ja testauksen yhdistämisessä. Suurin osa oli vielä POC-vaiheessa (*eng*. *Proof of Concept*) mutta joitakin ratkaisuja oli myös asiakkaiden käytössä. Haastateltavan H1 organisaatiossa ei hyödynnetä tekoälyä testauksessa, mutta hän kertoi projekteista, joissa hän on ollut mukana uransa aikana.

> H1: Siinä on tehty sellaista ratkaisua, jossa automaatio kattoo Jira-tikettiä, tai siis sun automaatiotestejä ennestään. Mutta koneäly tai tekoäly kattoo Jira-tikettejä bugien raportoinnin osalta. Ja yrittää tuottaa bugiraportille automaatiotestin, joka toistaa sen raportoidun virheen.

Haastateltava H1 kertoi, että testauksessa testitapaukset eivät ole se dokumentaatio, jota halutaan kirjottaa vaan esimerkit siitä, millä tavalla kyseinen ohjelmisto tai tapaus voi mennä pieleen. Puhutaan siis bugiraporteista. Kyseisessä kehitysprojektissa pyrittiin siihen, että kun asiakas raportoi tuotannossa olevan ongelman, niin tästä yritetään saada automaatio, joka toistaa saman ongelman. Kehittäjä näkee kyseisen ongelman omalla koneellaan automaation avulla ja pystyy testaamaan, tuliko se korjattua. Haastateltava H1 tarkensi, että projekti oli vielä tutkimusvaiheessa, mutta kohtuullisiin tuloksiin oli hänen mukaansa päästy.

> EC13: Tekoäly tuottaa manuaalisen virhekuvauksen perusteella automaattisen testin.

Haastateltava H1 korosti myös, että tekoälyn ja automaation avulla tulisi hoitaa ensisijaisesti helppoja virheluokkia, jotta ihmisten voimavaroja ei tarvitsisi käyttää ”perushygienian” ylläpitoon:

> H1: Perushygienia on sitä, että meillä on joitain luokkia virheitä, jotka on helpompi löytää kun toiset. Eli mä ite ajattelen usein niin, että mä ikään kuin slaissaan erilaisia virheluokkia, et meillä on jotain juttuja, jossa on helppoja sääntöjä. Et se hygienia on mulle sitä hyvän pohjan luomista. Jotta me voidaan sitten käyttää ne ihmisten voimavarat siihen, että ettei tartte pelkästään sitä hygieniaa hoitaa, vaan voidaan tehdä jotain joka tuottaa siihen päälle jotain lisäarvoa, jota kone yksinään ei pystyis hoitamaan.

Haastateltava H2 kertoi, että heillä myydään tekoälyä palveluna asiakkaan omaan ympäristöön, esimerkiksi tehtävänhallintaan, testaamiseen tai automaatioon. Tämä tarkoittaa sitä, että esimerkiksi testiautomaatiota täydennetään jollain tekoälykirjastolla tai algoritmilla. Haastateltava H2 kertoi tapauksen, jossa hyödynnettiin tekoälyalgoritmia ohjelmistorobotiikka-automaatioon. Asiakkaan täytyi vaihtaa lakisääteisiä tietoja muiden organisaatioiden kanssa sähköpostitse.

Sähköpostiviestit olivat eri kielisiä ja niitä lähetettiin ja vastaanotettiin tuhansia päivässä. Tämä vaati paljon henkilökunnalta manuaalista työtä, jonka vuoksi ohjelmistorobotiikka otettiin mukaan helpottamaan työtä. Automaatio kuitenkin hajosi usein, koska sähköpostiformaatti oli niin huono, jolloin viestit tulivat eri näköisinä, vaikka muoto oli sama.

Automaatioskriptiin lisättiin avoimen lähdekoodin tekoälykirjasto mukaan, joka tulkitsi viesteistä, mikä on riittävän oikein. Tekoäly osasi poimia oikeat asiat ja laittaa oikeaan paikkaan suomalaisen viraston tietokantaan. Loppujen lopuksi tekoälyalgoritmi osasi jopa toimia eri kielillä, vaikka se opetettiin yhdellä kielellä. Se kykeni hoitamaan esimerkiksi portugalinkielisen sähköpostin etsien riittävän samankaltaiset tiedot ja laittamaan viestit eteenpäin. Kyseisessä tapauksessa käytettiin ilmeisimmin NLP:tä, joskin haastateltava H2 ei ollut asiasta täysin varma, koska ei ollut tekemässä kyseistä projektia.

> H2: Sehän se onkin, me ei tehdä tekoälyratkaisuja vaan me tehdään ihan tavallisia ohjelmistorobotiikka ja testiautomaatioratkaisuja. Mutta niistä saadaan kestävämpiä, kun niihin käyttää oikeisiin paikkoihin tekoälyalgoritmejä. Eli niissä ei lue tekoäly oikeestaan missään. Ku niitä esitellään niin täs hyödynnettiin tekoälyä.

> EC14: Tekoälyä myydään palveluna asiakkaalle siten, että olemassa olevaa testiautomaatiota täydennetään tekoälykirjastoilla ja tekoälyalgoritmeilla.

Haastateltava H3 kertoi, että tekoälyavusteiseen käyttöliittymätestaukseen sekä testiajoon on meneillään projektit. Käyttöliittymäpuolella tutkitaan, saadaanko tekoäly tekemään käyttöliittymätestausta. Tässä tapauksessa tekoäly käy läpi käyttöliittymäpuuta ja jos se törmää virheeseen, se saa sisäisesti palkkion. Vahvistusfunktio tapahtuu, kun tekoäly menee useammin siihen kohtaan käyttöliittymästä, josta virhe löytyy. Tekoäly vertaa käyttöliittymää edelliseen versioon ja se saa palkinnon, jos se löysi eroavaisuutta. Algoritmi hyödyntää vahvistusoppimista ja konenäköä. Haastateltava H3:n mukaan projekti on POC-vaiheessa, mutta tulokset näyttävät hänen mukaansa hyviltä. Hän tarkensi, että tekoäly tutki vain yhtä käyttöliittymää ja algoritmin sisälle olisi hyvä saada muitakin käyttöliittymiä.

> H3: Voidaanko me heittää jotain meidän vanhaa testiautomaatiota pois, voidaanko sillä korvata jotain vai onko se jotain uutta jota meiltä kenties on puuttunut, tähän kysymykseen ei ole vielä vastausta mutta sen verran meillä on siitä kokemusta, että meillä niinkuin ainakin teknisessä mielessä toimii, mutta onko siitä semmosta konkreettista hyötyä, että meidän kannattaisi pitää sitä ajossa, niin siihen mä en vielä osaa vastata. Ehkä on tällainen paras vastaus.

Testiajoon liittyen haastateltava H3 kertoi, että pohdintaa on tehty, voisiko julkaisuputkesta syntynyttä dataa hyödyntää koneoppimisessa. Kun ohjelmisto julkaistaan ja pipeline hoitaa koko julkaisuputken, julkaisuputken data kertoo sen, miltä testitulosten tilanne sillä hetkellä näyttää. Haastateltava H3 tarkensi, voiko tähän kohti laittaa koneoppimista, jotta nähtäisiin muutoksia, jotka näyttäisivät liittyvän yhteen ja siitä seuraa jotain. Eli saataisiinko testiajo

ennustamaan, että tietyt testit kannattaisi ajaa ensin ja löytäisikö tekoäly uusia asioita ja ongelmia sovelluksesta. Tämä on haastateltava H3 mukaan kuitenkin vielä idea-asteella.

Haastateltavilla H4 ja H5 olivat havainneet, että testiautomaation tuottamia lokeja ja tiedostoja voisi tallentaa ja hyödyntää koneoppimismalleihin. Koneoppimismallit voisivat olla testimateriaalin valintaa ja luomista, sekä testitulosten analysointia. He totesivat, että ilman dataa ei voi olla koneoppimismallinnusta ja tästä syystä sellaistakin dataa olisi tärkeää kerätä talteen myöhempää käyttöä varten, jota testaajat eivät välttämättä miellä dataksi. Tämä voi olla kaikkea, mitä järjestelmä generoi, kuten onnistuneiden ja epäonnistuneiden testien tulokset sekä mitkä tekijät ovat aiheuttaneet nämä tulokset.

> H5: Ja sitten tavallaan se yks jäsennys mitä me (H4:n) kans miettiin kanssa semmoinen, että minkä tyyppistä dataa on ja mitä testaajat ehkä mieltää dataks? Että tavallaan se on yks mulla tavallaan oivallus kun mä oon vähän ulkopuolelta datatieteilijän näkökulmasta ja tietenkin se lähtöasetelma, että kaikki data voidaan olla hyödyllistä.

> EC15: Testiajojen tuloksista syntynyttä dataa voidaan hyödyntää koneoppimismalleihin.

Haastateltava H4 korosti erityisesti Intelligent Rulesien käyttöä. Haastattelija pyysi Haastateltava H4:ää avaamaan, mitä Intelligent Rulesit ovat.

> H4: Siinä ei ole mitään koneoppimista vaan se on niinku ifittelyä. Tavallaan niinku kone tekee, että jos on tämmöinen, niin tehdäänkin näin. Jos on taas toi toinen niin tehdään eri lailla, elikkä ne olisi semmoinen säännöstö, mitä ihminenkin orjallisesti noudattaa ja pystyisi pääsemään samaan tulokseen kuin kone. Mutta ne on vaan hirveän laajoja ja hirveän monimutkaisia, että niitä ei tehdä sen takia.

Haastateltava H4 totesi, että intelligent rulesin puolta ei kannata unohtaa. Siitäkin huolimatta, että se ei anna nopeita vastauksia vähällä työllä vaan vaatii toimiakseen paljon työtä ja nokkeluutta. Hän kertoi, että kun manuaalisesti analysoidaan asioita, niille voi tehdä sääntöjä, jotta kone tunnistaa ne esimerkiksi avainsanojen perusteella. Jos tapauksessa on virhekoodi ja integraationimi, tiettyjen ehtojen täytyttyä se analysoisi testitapauksen suoraan ja tulee tulokseen, jolloin suoritus päättyy, koska virhekoodiin liittyvät ehdot täyttyivät. Haastateltava H4 kertoi, että intelligent ruleseja on käytetty joissain organisaatioissa jo 20 vuotta sitten, mutta se ei ole kovin suosittu tapa.

Haastateltava H6 kertoi, että hän on ollut mukana projektissa, jossa testattiin testitapausten priorisointia tekoälyn avulla. Projektissa kerättiin analysoimisen avuksi suurten testimassojen historiadataa ja data tallennettiin tietokantaan. Historiadata ajettiin tekoälyalgoritmiin, jonka jälkeen virheitä saatiin poimittua paljon aikaisemmin, mitä normaalissa testiajossa. Tarkoituksena oli se, ettei ihmisen tarvitsisi valita itse joka ajolle uusia testejä, vaan tekoäly osaisi priorisoida kyseisen testiajon tärkeimmät testit, joista palaute halutaan saada nopeammin. Kyseinen ratkaisu ei ollut vielä käytössä asiakkaalla, vaan oli kokeiltu erilaisiin harrasteprojekteihin ja tehty tästä POC.

> H6: Kaikki testit ajetaan, se laadunvarmistus tehdään et se asia ei mee tavallaan eteenpäin, et sitä ei julkaista jos kaikki ei mee läpi tietyllä tavoilla. Toinen puoli on se et saatais se palaute niistä, et vaikka ne ajettas kaikki lopulta, mutta ajettas ne tärkeimmät alkuun et sais tavallaan, kun me tiedetään et suurin osa niistä regressioista ei feilaa niin antaa sen regression sit tapahtua myöhemmin ja yrittäs kuoria ne feilit sieltä regressiosta niinkun heti.

> EC16: Testitapausten priorisointiin on tehty POC, jossa tekoälyalgoritmi koulutettiin käyttäen testiajojen historiadataa.

Haastateltava H6 lisäsi, että pitkin viikkoa järjestelmään tulee muutoksia ja viikoittainen regressio sisältää usean koodaajan muutokset. Siksi olisi tärkeää saada kiinni ajoissa, mikä näistä monista asioista rikkoi kyseisen ominaisuuden. Olisi arvokasta pystyä poimimaan poikkeamat pieninä palasina päivittäin. Haastateltava H6 kertoi myös toisesta räätälöidystä ratkaisusta, joka oli tehty asiakkaalle. Ratkaisussa suoritetaan tuhansia testitapauksia fyysisille laitteille, jotka eroavat toisistaan, jolloin testitkin ovat keskenään erilaisia. Aikaisemmin ihminen on manuaalisesti joutunut jaottelemaan eri laitteille kuuluvat testit, mutta kyseisessä ratkaisussa tekoälyavusteinen automaatio osaa jakaa testit oikeille laitteille jokaisella ajolla.

Ratkaisu on säästänyt aikaa ja vaivaa poistaessaan fyysisten laitteiden manuaalisen testauksen. Haastateltava H6 kertoi, että aikaisemmin merkkausta oli käytetty testien jaottelussa, mikä johti siihen, että kukaan ei jaksanut muuttaa kovin tiuhaan merkintöjä, jolloin muutokset tehtiin vasta, kun huomattiin jonkun testiajon kestävän useita tunteja pidempään, kuin jonkun toisen. Älykkään automaation avulla laitteita voi muuttaa useita kertoja päivässä ja testiajo adaptoituu tähän joka ajolla.

> EC17: Tekoälyalgoritmi osaa tunnistaa erilaiset laitteet ja jaotella oikeat testit oikeille laitteille, vaikka laitteiden konfiguraatiot muuttuvat.

> EC18: Tekoäly on poistanut fyysisten laitteiden testien ylläpitovaiheen.

Haastateltava H7 kertoi, että heillä on ollut Test.ai:n valmis testausalusta evaluoitavana vajaan vuoden ajan käyttöliittymätestauksessa. Se ei ole vielä asiakasprojektissa käytössä. Haastateltava H7:n mukaan Test.ai ymmärtää käyttöliittymän konteksteja. Esimerkiksi suurennuslasi komponentissa tarkoittaa hakulaatikkoa. Test.ai:n algoritmille on syötetty erityisesti paljon verkkokauppoja, jolloin se osaa tunnistaa verkkokaupan käyttöliittymälle ominaiset asiat. Kun testiä ajetaan Test.ai:lla, niin käyttäjä on antanut suostumuksensa jakaa testattavan aineiston muille, jolloin harjoitusaineistoa karttuu Test.ai:n algoritmiin.

Haastateltava H7 pohti, että ajatus on hyvä, mutta Euroopassa, jossa noudatetaan GDPR:ää, eli EU:n tietosuoja-asetusta, datan keruu voi olla haastavampaa. Haastateltava H7 kertoi, että hänen yrityksessään on etsitty erityisesti verkkokauppa-asiakasta, jonka projektiin voisi Test.ai:ta ehdottaa kokeiltavaksi. Datan keruu ja säilytys huomioiden EU:n tietosuoja-asetus kuitenkin pohdituttaa haastateltava H7:ää.

Haastateltava H7 tarkensi, että Test.ai:ssa on vielä kehitettäviä asioita. Haastateltava H7 kuvaili Test.ai:n käyttöliittymän olevan hieman kömpelö. Lisäksi tuote on kehitetty verkkokauppa-alustoilla, jolloin sen avulla on haastavaa testata erilaisia alustoja. Se ei löytänyt oleellisia asioita haastateltava H7:n testaamasta sivusta eikä muodostanut loogista testimallia siitä, mitä haluttiin testata. Haastateltava H7 kertoi, että yrityksessä pyritään testaamaan ja kokeilemaan uusia ja erilaisia työkaluja testaukseen ja etsiä sovellusalueita, jossa niitä voisi käyttää.

> H7: Meillä on firmassa tällainen opintopiiri tai tällanen testaamisen tulevaisuuden tutkimusyksikkö. – – On tarkoitus näitä kaikkia työkaluja koeponnistaa, että me pystyttäisiin sit kertomaan siitä. Tai etsiä niitä sovellusalueita, missä näitä voisi oikeasti käyttää. Et aika pitkälti Suomessa testausfirmoja jos katsoo niin se on aika pitkälti se Robot Framework, mitä siellä käytetään. Niissä ei ihan hirveästi tehdä sellaista kokeilevaa tekemistä, että se tulee sitten lähinnä asiakasvetoisesti se juttu. Tää on kyllä sellainen, mitä me halutaan täällä edistää.

> EC19: Käyttöliittymätestaukseen on hyödynnetty valmista tekoälyratkaisua sekä räätälöityä koneoppimismallia.

Haastateltava H7 kertoi myös, että hän oli muutama vuosi sitten käynyt paljon Euroopassa konferensseissa. Tuona aikana tekoäly testauksessa oli vahvasti pinnalla oleva aihe ja tekoälypalveluja tarjoavia start-up firmoja oli paljon. Osa näistä on jo haastatteluhetkellä karsiutunut pois. Haastateltava H2 mainitsi myös samasta ilmiöstä.

Haastateltava H8 oli tehnyt opinnäytetyönään ohjelman, jonka avulla opinnäytetyön toimeksiantajan asiakkaat pystyivät tallentamaan kertaalleen tehdyn manuaalisen testin ja antaa automaation suorittaa testit jatkossa tallennukseen perustuen. Tässä hyödynnettiin pääasiassa ohjelmistorobotiikkaa ja testauskohteena oli raskaan liikkuvan kaluston HIL-testaus. Ohjelma lukee raskaan ajoneuvon dataa, analysoi sitä ja kirjoittaa testitapaukset Robot Frameworkiin esimerkiksi sen perusteella, onko kuljettaja painanut kaasua, miten ajoneuvon nopeus muuttuu tai istuuko kuljettaja penkillä. Ohjelmalla periaatteessa tarkistetaan, mitä ajoneuvossa tapahtuu ja sen avulla pystyttiin vertailemaan, että itse ajoneuvo ja ajoneuvon simulaattori toimivat samalla tavalla.

Ohjelmassa voidaan valita, mitkä signaalit ajoneuvossa halutaan tallentaa, jonka jälkeen signaalit tallennetaan samalla, kun ajoneuvo on käynnissä. Sen jälkeen tallennus lopetetaan ja luodaan Robot Framework testitapaukset. Signaalidata kerätään yhdistämällä kaapeli ajoneuvon ja kannettavan tietokoneen välillä. Opinnäytetyön tarkoituksena oli vähentää manuaalista testausta. Tulosten mukaan ohjelma generoi enemmän testiskriptejä Robot Frameworkiin manuaalisten testien perusteella.

Haastateltava H8 tarkensi, ettei ohjelma ole kuitenkaan täysin käytössä, koska se ei 100-prosenttisesti tee testausta. Haastateltava H8 tarkensi, että edelleen automaatiotestit kirjoitetaan manuaalisesti vaatimuksia mukaillen. Ohjelma siis kirjoittaa testiautomaatiota, mutta se kirjoittaa, mitä on tapahtunut traktorissa. Haastateltava H8 pohti, että tätä täytyisi vielä kehittää eteenpäin.

H8: No meillä oli ongelma, että ei ihan kaikin signaalit ole siellä. – – No ainakin tässä tapauksessa ihan traktorin HIL:iä varten se vähän kuin ei mennyt eteenpäin, mutta jossakin muussa projektissa se voisi olla varmaan hyödyllinen. Tässä traktorissa se vain oli kokeiltu ja tehty demo.

PEC4: Tekoälyä on hyödynnetty käyttöliittymätestaukseen, manuaalisen virhekuvauksen toistoon, testitapausten priorisointiin ja testiautomaatioon.
PEC5: Testiajoista syntynyttä dataa voidaan hyödyntää koneoppimisalgoritmien kehittämisessä. Koneoppimista, konenäköä, ohjelmistorobotiikkaa ja luonnollisen kielen prosessointia hyödynnettiin ratkaisuissa.

6.4.3 Tekoälyn hyödyntämisen haasteet

Haastateltavilta kysyttiin seuraavaksi, minkälaisia haasteita tekoälyn hyödyntämisessä testaukseen on tunnistettu. Kysymyksellä haluttiin selvittää, onko tekoälyn hyödyntämiselle testaukseen erityisiä esteitä, millaisia ne ovat ja voiko niihin vaikuttaa. Viisi haastateltavaa korostivat datan hyödyntämisen olevan esteenä tekoälyn hyödyntämisessä testaukseen. Koneoppiminen tarvitsee monipuolista harjoitusdataa, jotta se voi oppia toimimaan halutulla tavalla. Haastateltavat kertoivat, että testiajot tuottavat erilaisia lokitiedostoja ja raportteja, joita olisi hyvä säilöä tekoälyalgoritmin kehitystä varten. Haastateltavat H4 ja H5 totesivat, että datan hyödyntämisen mahdollisuuksia ei heidän organisaatiossaan tunnisteta. Lisäksi käyttökelpoista dataa heitetään säännönmukaisesti pois, koska dataa ei mielletä hyödylliseksi. Haastateltavat H4 ja H5 myös kertoivat, että ennen kuin koneoppimismalleja lähdetään rakentamaan, tulisi ymmärtää, että dataa voidaan hyödyntää jo perinteisillä menetelmillä. Dataa tulisi tutkia ja katsoa, mitä datasta löytyy.

H4: Jos mentäisiin kyselemään sitä, että miten tässä nyt voitaisiin hyödyntää koneoppimista ja mitä dataa sä siihen käyttäisit, niin testiautomatisoijat tänä päivänä hyvin todennäköisesti sanos, että no ei tässä oikein ole semmoista, että mitä voit hyödyntää, eikä mulla ole eikä ole semmoista dataa. Mutta se ei tarkoita todellakaan sitä, että sitä dataa ei olisi. Sitä ei vaan niinku nähdä ja miellä.

H5: Vaikka olisi resurssit ja tai jotakin että ei ole vielä se data. Niin sitten se käytännössä tarkoittaisi, että se jonkun tietyn POC:in aloittaminen viivästyy, vaikka muutamalla kuukaudella ja puolella vuodella. Tai jos nyt päättää sitten aloitetaan POC, niin oikeasti se voi alkaa vasta sen jälkeen, kun on tietty aika kerätty sitä dataa. Niin se voi olla merkittävä hidaste myös siinä.

Ongelmaksi haastateltavien mukaan muodostuu usein se, että historiadataa tulisi olla jo olemassa ennen kuin älykkäämpiä ratkaisuja testaukseen lähdetään toteuttamaan. Haastateltava H6 totesi, että kahden viikon takaista testiraporttia ei nykyisessä hetkessä pidetä arvokkaana, toisin kuin viimeisimpien testiajojen dataa. Tekoäly voisi esimerkiksi verrata kahden viikon takaista testiraporttia

ohjelmiston tietystä versiosta tämänpäiväiseen ja analysoida, mitkä asiat ovat muuttuneet.

> H6: Ne viime viikon tulokset on arvokkaita myöhemminkin. Tavallaan miten niitä kerätään ja sitten niihin liittyy sen laatu eli se että onko ne, tavallaan et on riittävä metadata. Se on itse aika usein se juttu näissä, että voidaan kerätä dataa. Mutta onko se riittävä metadataa, minkä perusteella se voidaan esimerkiksi yhdistää, että voidaanko me esimerkiksi yhdistää ne tulokset?

Haastateltava H1 näki datan ongelmallisuuden erityisesti datan monipuolisuudessa. Testauksessa tulisi pystyä testaamaan hyvin harvinaisiakin skenaarioita ja ilman monipuolista testidataa, mahdollisia virheitä ei löydetä. Data voi olla myös monen organisaation omistuksessa, jolloin siihen pääsy on hankalampaa.

> H1: Mut et tavallaan niinku tää data sille testaukselle on monellakin tapaa ongelma, kun sä et nää virheitä jos et sä saa sitä dataa monipuolistettua.

> EC20: Koneoppimisalgoritmin kehittämiseen tarkoitettua dataa ei tunnisteta eikä sitä kerätä tarpeeksi.

Testaustiimin, asiakkaiden ja organisaation valmius ottaa käyttöön älykkäämpiä ratkaisuja todettiin myös merkittäväksi pullonkaulaksi. Haastateltava H6 kertoi, että joissakin organisaatioissa vasta rakennetaan sitä infrastruktuuria, joka voisi mahdollistaa tekoälyavusteisia ratkaisuja. Monissa projekteissa jo olemassa olevien prosessien tulisi muuttua ennen uuden kehittämistä. Testiautomaatiota tulisi lähteä rakentamaan ennen kuin voidaan lähteä jatkojalostamaan testaista älykkäämmäksi. Tällaisissa tilanteissa testaustiimi ei ole ollut vielä kykenevä käyttämään tekoälyä hyväksi testauksessa.

> H6: Tärkein ongelma on se, et pitäs muuttaa olemassa olevia prosesseja ja tehdä sinne jotain, että jotta näistä vois hyötyä, niin täytys tavallaan tehdä lisää tai erilaisia asioita, johon yleensä ei löydy aikaa. Varsinkin sillon, jos puhutaan noista tuotejutuista, siellä on se, että jos siellä asiat toimii, niin on yleensä aika suostutteleminen saada ihmiset johonkin toimivaan koskea, et miten me tehtäs, ajateltas tää ihan eri tavalla.

Haastateltava H3 pohti, miten hänen nykyinen testiautomaatioinfrastruktuurinsa ja uusi tekoälytyökalu sulautuvat yhteen.

> H3: Sit kun sitä käyttää, niin minkälaisia ongelmia tai bugeja siitä tekoälyrutiinista tai siitä työkalusta jota käyttää, niin minkälaisia löytyy ja kuinka niitä sitten ratkaistaan ja tämmöstä. Niinkun normaalia softankehityshässäkkää. Ehkä ennemminkin ei tää silleen niinkun eroa siitä, elefanttia palastellessa pieniin palasiin ja sitä syödessä.

Kuusi haastateltavaa totesi merkittäväksi haasteeksi sidosryhmien asenteet tekoälyratkaisuja kohtaan. Sidosryhmillä tarkoitetaan pääasiassa asiakkaita, mutta myös haastateltavien omissa organisaatioissa on myös haasteita ratkaisujen kehittämiseen liittyen. Haastateltavat kuvailivat, että tekoälyn ottaminen

testauksen tueksi voi työllistää hetkellisesti enemmän testaustiimiä, mutta todennäköisesti helpottaa työkuormaa tulevaisuudessa. Tämä on haastateltavien mukaan koettu liian vaivalloiseksi testaajien ja asiakkaiden osalta, jolloin kokeiluihin ei ole haluttu lähteä. Haastateltavien H4:n, H5:n ja H6:n johtopäätökset olivat datan keruuseen sekä testaustiimin valmiuteen liittyen samat.

Neljä haastateltavaa mainitsi, että asiakkaat eivät ole olleet valmiita kokeilemaan tekoälyavusteisia ratkaisuja, koska aikaisempia referenssejä ei ole. Tässä päädytään tilanteeseen, jossa ensimmäistäkään julkista referenssiä on hankala saada, jos aikaisempiakaan ei ole. Haastateltavat H4 ja H5 kuvailivat, että organisaatiossa ei olla valmiita investoimaan tuleviin hyötyihin ja investoimaan omakustanteisesti sisäisiin projekteihin aiheen parissa, jotta asiakkaita olisi helpompi saada.

> H4: Me itse kehitettäisi jotain ratkaisuja, jotka me voitaisiin, kun meillä olisi ne ratkaisut. Me voitaisiin myydä ne asiakkaille. Tässä on tietenkin nyt se myös, että tällä hetkellä kun meillä on asiakkaita, niin asiakkaan ei ole sillä lailla valmiita ostamaan asioita, jotka kehitettäisiin heillä. Että ei tällä hetkellä juuri löydy semmoisia organisaatioita. Oma organisaatio mukaan lukien, joka siellä omalla rahalla kehittäisi sitä tulevaisuutta. Mä näen, että olisi noita, jotka rahoittaa sitä, että ihan tietäen sitä, että me kokeillaan erinäköisiä asioita. Osa niistä toimii. Osa ei tule toimimaan, mutta yksittäiset yritykset tuntuu aika nihkeästi haluavan ottaa semmoista riskiä.

Haastateltava H2 kuvaili ongelmaa seuraavasti:

> H2: Et halutaanko myydä uusia tehokkaampia ajatuksia, sitä ollaan myyty, yllättävän vähän sitä on mennyt kaupaksi. Hirveän monella asiakkaalla on intoa, mutta kovin moni ei sit halua lähteä eteenpäin meidän kanssa. Paljon on sitä "me ollaan totuttu tekee näin, meillä on hyvät testaajat, ne osaa priorisoida", menee nykyisillä tavoilla liian usein. – – Mut aika pientä se meillä vielä on et voidaan sanoa, että se on meillä vielä kokeiluasteella, kuten sanottua, asiakkaat haluaa vaan hyvää testiautomaatioo, ne ei haluu tekoälyä, ne aattelee et se on jotain ihan uutta, viittiiks tommoseen panna rahaa, et ei ne haluu siihen erikseen investoida. Et asiakkaat haluaa vaan hyvää ja hyvän hintasta testiautomaatioo.

Haastateltava H2 kertoi, että hänen organisaatiossaan on tehty tekoälykokeiluja omakustanteisesti, jos asiakas ei ole ollut valmis maksamaan. Haastateltava H2 perusteli asiaa niin, että tekoälyn osuus testiautomaatioprojekteista kasvaa jatkuvasti, jolloin organisaation tulee pysyä kehityksessä mukana, Muuten hävitään kilpailu asiakkaista jossain kohtaa. Tekoäly on myös koettu jopa pelottavana tai vaikeana asiana, jos tekoälyn algoritmia ei ymmärretä. Haastateltava H2 korosti, että olisi hyvä, että tekoälyä ei mystifioitaisi, vaan sen voisi vain ottaa käyttöön yhtenä komponenttikirjastona, joka lisätään Robot Framework ympäristöön. Hänen mukaansa pullonkaula tekoälyratkaisujen yleistymiseen on se, että se koetaan liian vaikeaksi, jolloin siitä pitäisi tehdä helpommaksi ymmärtää.

> EC21: Asiakkaat eivät ole valmiita kokeilemaan tekoälyratkaisuja testaukseen. Organisaation olemassa olevat prosessit eivät ole vielä tarpeeksi kehittyneitä tekoälyn hyödyntämistä varten.

Haastateltava H8 pohti haasteita tekoälyn hyödyntämisestä vaatimustenmäärittelyyn. Hän totesi, että testitapausten kirjoittaminen vaatimusten mukaan olisi haastavaa tekoälyn toimesta, koska ihmisten tekemänä vaatimusten kirjoittaminen on myös hankalaa, eivätkä vaatimukset ole aina ymmärrettävässä muodossa.

> H8: Jos me halutaan, että se kirjoittaisi testit sitten kaiken vaatimukseen määrittäminen selvällä kielellä. Koska monesti edes ihan kaikki vaatimukset eivät ole kirjoitettu. Pitää vain pitää muistaa, että nyt pitäisi olla näin, mutta se ei hirveästi esimerkiksi dokumentoida. Sellaista ei saa, että olisi ollut kaikki ihan selvästi.

PEC6: Tekoälyyn tarvittavaa dataa ei kerätä, eikä sitä tunnisteta. Asiakkailla ja haastateltavien organisaatiolla ei ole valmiutta tai halukkuutta ottaa tekoälyavusteista testausta käyttöön.

6.4.4 Tekoälytestauksen tulevaisuus

Haastateltavilta kysyttiin lopuksi, millaisia jatkosuunnitelmia organisaatiolla on tekoälyn hyödyntämiseen testauksessa ja miten haastateltavat näkevät aiheen tulevaisuuden. Jatkosuunnitelmat tulivat osittain ilmi jo edellisestä kysymyksestä, mutta osa haastateltavista pohti tekoälyn nykytilaa testauksessa ja mitä se on tulevaisuudessa. Kysymyksellä haluttiin saada käsitystä siitä, kohtaavatko kirjallisuuskatsauksen pohjalta syntyneet arviot tekoälyn yleistymisestä haastateltavien pohdintojen kanssa.

Neljä haastateltavaa totesi, että asiakastapauksia tulisi löytää lisää, jotta tekoälyn kehitystä testaukseen voidaan laajentaa. Kehitys junnaa monessa tapauksessa paikallaan, koska asiakkaat eivät uskalla lähteä kokeilemaan uusia ratkaisuja, ennen kuin referenssejä on muistakin tapauksista. Haastateltavat H4 ja H5 kertoivat, että heillä pyritään luomaan markkinointiviestintää aiheen ympärille videoiden ja webinaarien muodossa, joiden avulla pyritään saamaan potentiaalisia asiakkaita innostumaan aiheesta.

> H5: Ehkä se siellä hakee sitä, että mikä se asiakas POC olis, että se on nyt vähän ehkä tyhjäkäynnillä tällä hetkellä. Et siinä ei oo konkretiaa, että se on ehkä enemmänkin, että yritetään luoda sitä markkinointiviestiä. Tällaista on kehitteillä ja sitten halutaan pitää se sillä tavalla näkyvissä ja pystyssä, mutta tavallaan se konkretia ehkä vähän vielä puuttuu.

> EC22: Tekoälyavusteiseen testaukseen tulisi löytää asiakkaita, jotta kehitystä päästään jatkamaan.

Useampi haastateltava totesi, että tekoäly on jo monessa arkipäiväisessä työkalussa mukana ilman, että käyttäjä huomaa käyttävänsä sitä. Haastateltava H4 sanoi, että testidatan generointi ja testitulosten analysointi todennäköisesti kehitetään Azuren pilvipalveluun ja käyttäjältä kysytään, haluaisitko luoda automaattisesti testidataa itsellesi. Tällä tavalla tekoäly arkipäiväistyy yhä enemmän, eikä sitä mystifioida.

Haastateltava H1 pohti, että ohjelmistotyökalut ovat kehittyneet jo niin paljon ilman tekoälyä, että jo kehitystyössä palaute ohjelmiston laadusta saadaan nopeammin. Tämän seurauksena testaajien osuus kehitystiimistä on pienentynyt.

> H1: Tää on se tuleva trendi, jonka uskon tapahtuvan yhä enemmän, et se säästää testausaikaa kehittäjiltä, se säästää testausaikaa testaajilta.

Haastateltava H6 toivoisi, että jo suunnitellut toimenpiteet tekoälyn hyödyntämisestä testaukseen saataisiin vietyä käytäntöön ja jalkautettua muihinkin projekteihin. Hän totesi, että jo suunnitellut projektit palaavat datankeruuongelmaan ja ajankäytölliset resurssit eivät ole vielä riittäneet siihen, että toteutusta lähdettäisiin työstämään. Haastateltava H7 toivoi, että tämänhetkisestä akateemisesta tuotekehitysvaiheesta päästäisiin kuluvan vuoden aikana todelliseen tuotantokäyttöön. Sitten tiedettäisiin paremmin, millaiselle asiakkaalle tarjota käyttöliittymätestausta ja kuka sitä haluaisi kokeilla. Keskusteltaessa tekoälytestauksen tulevaisuudesta, haastateltava H7:n pohdinta oli hyvin samansuuntaista, kun haastateltava H1:llä.

> H7: Ohjelmistokehittäjän rooli on muuttunut sinä aikana, kun itsekin alalla aloitti niin roolit oli hyvin segrekoitu sillain, että jotkut koodas, jotkut asensi, sitten jotkut käänsi sitä koodia. Nyt kun sä oot softadevaaja niin sun pitää tehdä se kaikki, sun pitää osata vielä niinku pilviarkkitehtuuri. Kaikki se tavallaan se tekninen ympäristö on muuttunut tosi paljon. Että sama koskee testausta. Ei se niinku nyt testausta ammattina poista, mutta se rooli varmaan välillä muuttuu.

Haastateltava H8 kertoi, että HIL:iin kehitetyn testiautomaatiodemon jälkeen tällä hetkellä ei ole suunnitteilla tekoälyn hyödyntämisratkaisuja.

> EC23: Tekoäly on jo monessa arkisessa teknologiassa mukana, mutta tekoäly ei tule vielä korvaamaan testaajien työtä.

> PEC7: Tekoälyratkaisuja saataisiin tulevaisuudessa kehitettyä paremmin, jos asiakkaita saadaan mukaan kehitykseen. Tekoäly arkipäiväistyy yhä enemmän, mutta ei vielä korvaa nykyisten testaajien työtä.

6.5 Yhteenveto

Kappaleessa 6 käytiin läpi tutkimuksen tulokset. Aineiston perusteella muodostettiin 22 empiiristä johtopäätöstä ja 7 pääasiallista johtopäätöstä.

TAULUKKO 2 Tutkimuksen empiiriset johtopäätökset

Tunniste	**Empiirinen johtopäätös**
EC1	Tekoälyn älykkyys on ongelman ratkaisua ja päätöksentekoa perustuen annettuun dataan.

EC2	Tekoälyn älykkäät toiminnot perustuvat koneoppimiseen.
EC3	Testaus ja ohjelmistokehitys eivät ole erillisiä toimintoja, vaan testaus on osa ohjelmistokehitystä.
EC4	Testaus tulisi sisällyttää ohjelmistokehitykseen alusta alkaen.
EC5	Testiautomaation avulla varmistetaan, että ohjelmiston ominaisuudet eivät hajoa kehityksen aikana.
EC6	Testiautomaatio ja manuaalinen testaus ovat rinnakkaisia, eivätkä toisistaan erillisiä toimintoja.
EC7	Tutkivalla testauksella pyritään löytämään ohjelmistosta ne viat, joita testiautomaation avulla ei löydetä.
EC8	Testiajojen tuottaman palautteen saanti kestää liian kauan.
EC9	Testitapausten suorittamisjärjestystä tulisi voida priorisoida.
EC10	Testiautomaatio ei mukaudu ohjelmiston muutoksiin itsenäisesti.
EC11	Käyttöliittymätestit hajoavat helposti ja testien ylläpidettävyys on työlästä.
EC12	Testidokumentoinnin luonti ja vaatimusten määrittely koettiin työlääksi.
EC13	Tekoäly tuottaa manuaalisen virhekuvauksen perusteella automaattisen testin.
EC14	Tekoälyä myydään palveluna asiakkaalle siten, että olemassa olevaa testiautomaatiota täydennetään tekoälykirjastoilla ja tekoälyalgoritmeilla.
EC15	Testiajojen tuloksista syntynyttä dataa voidaan hyödyntää koneoppimismalleihin.
EC16	Testitapausten priorisointiin on tehty POC, jossa tekoälyalgoritmi koulutettiin käyttäen testiajojen historiadataa.
EC17	Tekoälyalgoritmi osaa tunnistaa erilaiset laitteet ja jaotella oikeat testit oikeille laitteille, vaikka laitteiden konfiguraatiot muuttuvat.
EC18	Tekoäly on poistanut fyysisten laitteiden testien ylläpitovaiheen.
EC19	Käyttöliittymätestaukseen on hyödynnetty valmista tekoälyratkaisua sekä räätälöityä koneoppimismallia.
EC20	Koneoppimisalgoritmin kehittämiseen tarkoitettua dataa ei tunnisteta eikä sitä kerätä tarpeeksi.
EC21	Asiakkaat eivät ole valmiita kokeilemaan tekoälyratkaisuja testaukseen. Organisaation olemassa olevat prosessit eivät ole vielä tarpeeksi kehittyneitä tekoälyn hyödyntämistä varten.
EC22	Asiakastapauksia tekoälyavusteiseen testaukseen tulisi löytää, jotta kehitystä päästään jatkamaan.
EC23	Tekoäly on jo monessa arkisessa teknologiassa mukana, mutta tekoäly ei tule vielä korvaamaan testaajien työtä.

Empiiristen johtopäätösten perusteella muodostettiin seitsemän pääasiallista johtopäätöstä, joiden pohjalta tutkimuksen keskusteluosuus muodostetaan.

TAULUKKO 3 Tutkimuksen pääasialliset empiiriset johtopäätökset

Tunniste	**Pääasiallinen empiirinen johtopäätös**
PEC1	Tekoäly voi ratkaista ongelmia ja tehdä päätöksiä annetun datan perusteella. Tekoälyn toiminnot perustuvat koneoppimiseen.
PEC2	Ohjelmistotestaus on osa ohjelmistokehitystä. Testiautomaatio toimii ohjelmiston laadun turvaverkkona. Tutkivalla testauksella pyritään löytämään ne ominaisuudet, joita testiautomaation avulla ei löydetä.
PEC3	Testauksessa koettuja haasteita ovat testiajoista saadun palautteen viive, käyttöliittymätestien ja testiautomaation ylläpidettävyys sekä testidokumentoinnin tuottaminen manuaalisen kuvauksen perusteella.
PEC4	Tekoälyä on hyödynnetty käyttöliittymätestaukseen, manuaalisen virhekuvauksen toistoon, testitapausten priorisointiin ja testiautomaatioon.
PEC5	Testiajoista syntynyttä dataa voidaan hyödyntää koneoppimisalgoritmien kehittämisessä. Koneoppimista, konenäköä, ohjelmistorobotiikkaa ja luonnollisen kielen prosessointia hyödynnettiin ratkaisuissa.
PEC6	Tekoälyyn tarvittavaa dataa ei kerätä, eikä sitä tunnisteta. Asiakkailla ja haastateltavien organisaatiolla ei ole valmiutta tai halukkuutta ottaa tekoälyavusteista testausta käyttöön.
PEC7	Tekoälyratkaisuja saataisiin tulevaisuudessa kehitettyä paremmin, jos asiakkaita saadaan mukaan kehitykseen. Tekoäly arkipäiväistyy yhä enemmän, mutta ei vielä korvaa nykyisten testaajien työtä.

7 KESKUSTELU

Kappaleessa pohditaan, miten tutkimuksen pääasialliset johtopäätökset linkittyvät olemassa olevaan teoriaan ja millaisia käytännön implikaatioita johtopäätöksistä voidaan muodostaa. Käytännön implikaatiot kiteyttävät tutkimuksen todelliset johtopäätökset. Johtopäätöksiä voidaan hyödyntää käytännön työhön tekoälyn hyödyntämisessä testaukseen.

7.1 Teoreettiset implikaatiot

PEC1 kiteyttää tekoälyn määritelmän empiirisen aineiston perusteella. Olennaista tekoälylle on, että se voi ratkaista ongelmia ja tehdä itsenäisiä päätöksiä annetun datan perusteella. Kayidin (2020) ja Shin (2011, s. 9–10) määritelmät sisältävät PEC1:n asiat, joskin teoria-aineistossa käsitellään tekoälyä tieteenalana, ei yksittäisenä toimijana. Shin (2011 s. 9–10) määritelmän mukaan tekoälytutkimuksessa kehitetään inhimillisen käyttäytymisen laskennallisia malleja järjestelmiin, jotka havainnoivat, päättelevät, oppivat, assosioivat, tekevät päätöksiä ja ratkaisevat monimutkaisia ongelmia ihmisen tavoin. Haastateltavista kaksi toivat esiin tekoälyn inhimilliset kyvyt, mutta pääosin tekoäly koettiin ongelmanratkaisijana. Koneoppimisen katsotaan olevan iso osa tekoälyä teoria-aineiston perusteella, mutta koneoppiminen erotetaan olevan tekoälyn alalaji. Haastateltavat totesivat tekoälyn olevan yhtä, kuin koneoppiminen.

Manuaalisen ja testiautomaation eroa korostettiin kirjallisuudessa paljon. Teoria-aineistossa käsiteltiin ohjelmistotestausta varsin yksittäisenä elementtinä, joskin ohjelmistokehityksen V-mallissa korostetaan testauksen olevan jatkuvaa, rinnakkaista toimintaa ohjelmistokehityksen kanssa. PEC2 kuitenkin osoittaa, että testauksessa ei ole välttämättä yksittäistä prosessia, vaan se etenee vahvasti ohjelmiston kehityksen kanssa. Manuaalinen ja testiautomaatio ei nähty myöskään vastakkainasettelun kautta. Haastateltavat painottivat, että testit tulisi pystyä automatisoimaan, jos mahdollista, mutta testiautomaatio ei kaikkeen ohjelmistotestaukseen taivu. Kuten Graham & Fewster, (2012) totesivat, automaatio

on pääasiassa keino ajaa isoja joukkoja testejä, eikä se ota kantaa siihen, testataanko oikeita asioita. Testiautomaation avulla testaukseen saadaan laajuutta ja kuten Goericke (2020) toteaa, ohjelmistotestausta nopeutetaan merkittävästi, jolloin testiautomaatiolla on iso vaikutus ohjelmistokehityksessä. Haastateltavat totesivat tutkivan testauksen olevan manuaalista testausta, koska sen avulla ohjelmistoa tutkitaan syvällisemmin ja testaaja ikään kuin keskustelee ohjelmiston kanssa. Tutkivan testauksen roolia ohjelmistotestauksessa ei kirjallisuudessa korostettu.

Santiagon ym. (2018) mukaan testiautomaatio ei voi skaalautua ohjelmistojen välillä eikä jäljittelemään ihmisälykkyyttä. Amannejad ja Garousi (2014) toteavat, että automaattisen testauksen toteuttaminen, ylläpito sekä epäonnistuneiden testien analysointi vievät testaajilta aikaa. PEC3 osoittaa testiautomaation ylläpidon yhdeksi testauksen tämänhetkisistä haasteista, erityisesti käyttöliittymätestauksessa. Volk (2021) väittää, että testaustiimit kirjoittavat ja päivittävät koodia useista hajautetuista mikropalveluista koostuviin sovelluksiin ja ohjelmistoihin, jonka vuoksi testauksesta tingitään ensimmäisenä. (Volk, 2021.) PEC3 osoittaa, että haasteet ovat testiajojen palautteen viiveessä, jolloin testituloksista täytyisi saada tietoa nopeammin, erityisesti kriittisten testien osalta. Haastateltavat korostivat, että kaikkea ei voi testata, joten testejä tulisi voida priorisoida. Yksi haastateltavista sanoi, mitä hajautetumpi ohjelmisto on, sitä enemmän siinä tulisi olla älykkyyttä mukana.

Volk (2021) korostaa tekoälyn hyödyntämiskohteeksi testitapausten priorisoinnin, eli tekoälyn avulla testausta voidaan tehostaa karsimalla ne testitapaukset, jotka eivät vaadi testaajan huomiota. Muita kohteita ovat testauksen kokonaistyönkulun automatisointi, inhimillisten tehtävien yhdistäminen, suositusten tarjoaminen testaajille ja oppiminen ihmisten tekemistä päätöksistä. (Volk, 2021.) Kirjallisuudessa testauksen haasteista tuotiin esiin manuaalinen koodin kirjoittaminen ja monien toiminnallisuuksien testaaminen, kuten Bellapu (2021) kiteyttää. PEC3 osoittaa haasteet hienojakoisemmin, mihin konkreettiseen toiminnallisuuteen haasteet kulminoituvat.

PEC4 osoittaa ne testauksen osa-alueet, johon tekoälyä on lähdetty tutkimustapauksissa hyödyntämään. Testitapausten priorisointi ja käyttöliittymätestaus nousivat aineistossa yleisimmiksi tekoälyn hyödyntämiskohteiksi. Yksi haastateltavista käytti Test.ai:ta käyttöliittymätestaukseen. Santiagoa (2018) mukaillen, Test.ai on ollut isossa roolissa tekoälypohjaisten testausteknologioiden kehityksessä. Se hyödyntää sillä testattujen sovelluksien testidataa ja käyttää sitä uudelleen luodakseen yleisiä testitapauksia. Sen teknologiassa hyödynnetään vahvistusoppimista ja konenäköä testitapausten luontiin. (Santiago ym., 2018; Test.ai, 2022.) Myös Volk (2021) toteaa, että visuaalisessa testauksessa on tällä hetkellä eniten valmiita ratkaisuja. Yksi haastateltavista kertoi myös, että käyttöliittymätestaukseen on kehitteillä POC. Molemmissa tapauksissa haastateltavat totesivat, että tekoälyn olisi hyvä käydä läpi erilaisia käyttöliittymiä. Test.ai:n haasteeksi muodostui sen yksipuolisuus, koska sen tekoälyalgoritmit ovat oppineet testaamaan pääosin verkkokauppoja. Visuaalinen testaus on kasvava alue tekoälyavusteisessa testauksessa, mutta aineiston perusteella se vaatii vielä

monipuolisempaa harjoitusdataa pystyäkseen testaamaan erilaisia ohjelmistoja hyvin.

Yksi haastateltava kuvaili kokeiluprojektia, jossa testiajojen historiadataa käytettiin tekoälyalgoritmin opettamiseen. Kaksi haastateltavaa olivat tulleet myös samaan johtopäätökseen testiajojen historiadatan hyödyntämisestä. Spiekerin ym. (2017) tutkimuksessa esiteltiin koneoppimisalgoritmi, joka pyrkii valitsemaan ja priorisoimaan testitapaukset niiden suoritusympäristössä. Priorisoituja testitapauksia on käytetty onnistuneesti vikojen havaitsemiseen ohjelmiston jatkuvan integraation sykleissä. Lupaavimmat testitapaukset ajetaan ensin. Vaikka tarve testitapausten priorisoinnille on suuri, konkreettisia ratkaisuja ei ollut haastateltavien mukaan tehty, yhtä lukuun ottamatta. Tämä kokeilu tehtiin harrasteprojektille.

Bellapu (2021) kuvailee testauksen automatisointia siten, että testaaja kirjoittaa koodiskrptin kohteen toimivuuden testaamiseksi. Kun testi on todettu toimivaksi, koodia voidaan käyttää uudelleen muutosten testaukseen, jolloin testauskoodin uudelleenkirjoittaminen eliminoituu. (Bellapu, 2021.) Kaksi haastateltavaa kertoivat testiautomaation rikastamisesta tekoälyn avulla. Toinen heistä kuvaili ratkaisua, jossa tuotannosta tulleen manuaalisen virhekuvauksen perusteella tekoäly luo automaattisen testin, joka toistaa kyseisen virheen. Tällaiseen menetelmään ei teoria-aineistosta löytynyt vastaavaa tapausta. Bellapun (2021) kuvailema esimerkki oli toteutettu tekoälyn avulla.

Sulautettujen järjestelmien testaukseen oli tehty kaksi esimerkkiä. Tekoälyn avulla testit on saatu jaoteltua keskenään erilaisille laitteille riippumatta siitä, onko laitteisiin tehty muutoksia. Testit on jouduttu aikaisemmin jaottelemaan manuaalisesti, mutta tämä työvaihe on tekoälyn myötä poistunut. Toisessa tapauksessa liikkuvan kaluston manuaaliset testit tallennetaan ja RPA:n avulla luo Robot Framework -testit. Testit ajetaan regressiotesteinä jatkossa, jolloin manuaalinen vaihe poistuu. Fyysisten laitteiden ja sulautettujen järjestelmien testauksesta ei löytynyt teoria-aineistosta esimerkkejä, mutta jälkimmäinen tapaus kuvastaa myös Bellapun (2021) esimerkkiä koodin uudelleen käytettävyydestä. Ero on siinä, että ratkaisun koeajon nauhoitus luo Robot Frameworkiin testit automaattisesti. Haastateltava totesi, että ratkaisua tulee kehittää eteenpäin, koska ratkaisu kirjoittaa testiautomaatiota siitä, mitä liikkuvassa kalustossa on tapahtunut.

PEC6 kiteyttää haasteet tekoälyn hyödyntämisen käyttöönotolle. Kuten Neittaanmäki ja Tuominen (2019) toteavat, tekoälyalgoritmit eivät voi luoda oppimaansa malliin tietoa, joita niiden opettamiseen käytettävässä datassa ei ole. Tekoälyn laatu ja kattavuus on datasta riippuvaista, jolloin syötetystä datasta eroava tieto on tekoälylle vaikeaa käsitellä. Tekoälyalgoritmin opettaminen onnistuu siinä vaiheessa, kun monipuolista dataa on jo kerätty. PEC6 mukaan organisaatioissa on joko vaikeuksia löytää tarpeeksi rikasta dataa tai käyttökelpoista dataa ei tunnisteta.

Toinen haaste on tekoälyratkaisuiden käyttökohteiden löytäminen. Asiakkaiden nykyiset toimintatavat eivät ole tarpeeksi valmiita testiautomaation käyttöönottoon, jolloin tekoälyratkaisut ovat liian kehittyneitä. Asiakkaat eivät

myöskään miellä tarvitsevansa tekoälyä testauksen tehostamiseen, vaan testiautomaatio riittää. Ramchand ym. (2021) mukaan, tekoälyä ei ole hyödynnetty sen laajimmassa potentiaalissaan ohjelmistotestauksessa. Testausammattilaisilla ei ole tarpeeksi tietämystä, millä tavalla tekoälyalgoritmeja voidaan käyttää, mutta testiautomaatiotyökaluja osataan hyödyntää. Empiirisen aineiston perusteella voidaan päätyä samaan johtopäätökseen. PEC7 osoittaa ne asiat, joita organisaation tulisi tehdä, jotta tekoälyn kehitys testauksessa etenisi. Lisäksi keskusteltiin siitä, millaisena haastateltavat näkevät ilmiön tulevaisuuden. King ym. (2019) tutkimuksessa 23 % vastaajista 328 vastaajan joukosta oli sitä mieltä, että tekoälypohjainen testaus tulee korvaamaan manuaalisen testauksen noin 5 vuoden sisällä, kun taas 35 % oli sitä mieltä, ettei tekoäly tule korvaamaan koskaan täysin manuaalista testausta.

Kukaan haastateltavista ei suoranaisesti uskonut, että manuaalista testausta tullaan korvaamaan lähitulevaisuudessa, mutta tekoälyratkaisut tulevat arkipäiväistymään eri osa-alueilla, kuten pilvialustoilla. Santiago ym. (2018) kiteyttävät, että tekoäly- ja koneoppimistestausta tapahtuu jo nyt. Yritykset alkavat kiinnittää huomiota tähän osa-alueeseen etenevissä määrin ja tutkimus- ja kehitystyö tuottaa lisää dataa aiheen ympärille. (Santiago ym., 2018.) Hourani ym. (2019) on sitä mieltä, että tekoälypohjaisesta testauksesta tulee kokonaan itsenäinen ala ja tekoälytestaus korvaa ohjelmiston laadunhallinnan ammattilaiset ja testaajat. Santiagon ym (2018) väite on linjassa empiirisen aineiston kanssa, mutta mitä tulee autonomiseen testaukseen, haastateltavien mukaan tekoäly ei korvaa testaajien työtä vielä vuosiin. Tekoäly tulee keventämään ja nopeuttamaan ohjelmiston peruslaadun ylläpitoa ja rikastamaan testiautomaatiota. Kuten Ramchand ym. (2021) toteaa, tekoäly ei korvaa vielä manuaalista työtä tai ole kaiken kattava testaustyökalu.

7.2 Käytännön implikaatiot

Käytännön implikaatiot kuvaavat tutkimuksen todelliset tulokset. PEC1 osoittaa, että tekoäly voi ratkaista ongelmia ja tehdä päätöksiä data-aineiston perusteella. Tekoäly on tänä päivänä sama, kuin koneoppiminen. PEC2 osoittaa, että haastateltavien mukaan ohjelmistotestaus on osa ohjelmistokehitystä, eikä sitä voida irrottaa erilliseksi prosessiksi. Ohjelmistotestaus on sama asia, kuin ohjelmistokehitys, mutta kolikon toinen puoli. Automaattinen testaus ja manuaalinen testaus varmistavat ohjelmiston laatua eri tavoin. Manuaalisia testausvaiheita tulisi pyrkiä automatisoimaan, mikäli ne nopeuttavat ja helpottavat testausta. Hyvä testiautomaatio toimii ohjelmiston laadun turvaverkkona ja manuaalisella, tutkivalla testauksella pyritään löytämään ne virheet, joiden löytämiseen testiautomaatio ei taivu.

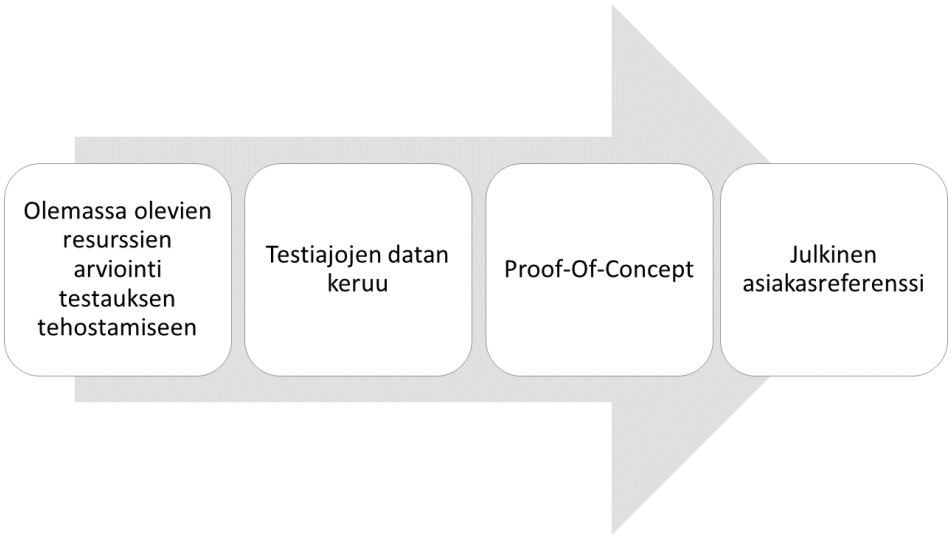
PEC3 mukaan suurimmat haasteet testauksessa ovat testiajojen palautteen saannin kesto, käyttöliittymätestien ja testiautomaation ylläpidettävyys. Testauksessa joudutaan myös tuottamaan dokumentaatiota, jolla ei välttämättä ole asiakkaan tai testauksen kannalta merkitystä, mutta joita laatustandardit voivat

vaatia. Suurin tarve tekoälylle on testitapausten priorisoinnissa ja käyttöliittymätestauksessa. Tekoälyltä toivottiin myös helpotusta vaatimusten määrittelyyn, eli tekoäly tekee manuaalisen vaatimusten perusteella testit. PEC3 ja PEC4 ovat linjassa, sillä PEC4 osoittaa, miten testauksen haasteisiin on vastattu tekoälyn ja koneoppimisen avulla. Käyttöliittymätestaukseen on jo olemassa valmiita ratkaisuja, mutta perustuen haastateltavien kokemuksiin, ne vaativat vielä harjoitusdataa erityyppisistä käyttöliittymistä toimiakseen hyvin.

Tekoälykirjastoja ja tekoälyalgoritmeja voidaan jo hyödyntää testiautomaation rikastamiseen ja tästä on saatu jo varsin hyviä tuloksia. Tekoäly oppii ympäristönsä datasta jatkuvasti ja testiajoista syntyneen datan avulla tekoälyalgoritmi voidaan opettaa priorisoimaan tärkeimmät testit ajettavaksi ensin. Tekoälyalgoritmin avulla on jopa saatu sähköpostikeskusteluja käännettyä eri kielille. PEC5 kuvailee, miten tekoälyratkaisuja on saatu kehitettyä. Tärkein materiaali tekoälyn toiminnalle on data. Datan avulla koneoppimisalgoritmi on oppinut toimimaan testiympäristössä. Ratkaisuissa hyödynnettiin koneoppimista, konenäköä, luonnollisen kielen prosessointia ja ohjelmistorobotiikkaa.

Seuraavaksi päästään PEC6 osoittamiin pullonkauloihin, joita tekoälyn hyödyntämisessä ilmeni. Testaukseen hyödynnettävään tekoälyalgoritmiin tarvittavaa dataa ei tunnisteta eikä sitä kerätä. Dataa tulisi olla jo olemassa siinä vaiheessa, kun tekoälyratkaisuja lähdetään kehittämään. Tekoälytestaus vaatii myös kehittyäkseen käyttökohteita, eli asiakasprojekteja. Asiakkaat eivät ole olleet vielä innokkaita kokeilemaan tekoälyratkaisuja ilman organisaation aikaisempia referenssejä.

Organisaatio voisi panostaa kehitystyöhön omakustanteisestikin, jotta asiakkaille saadaan tarjottavaksi älykkäämpiä ratkaisuja. Tekoälyratkaisut vaativat myös aluksi työtä ja vaivannäköä voidakseen helpottaa testaajien arkea tulevaisuudessa. Tämä vaatii testaustiimiltä ja organisaatiolta uskoa siihen, että tekoälyratkaisut tulevat helpottamaan työtä tulevaisuudessa. Tekoäly ilmiönä koetaan myös hankalaksi ymmärtää niin testaajien kuin asiakkaiden mielestä. Vaikka tekoälyalgoritmin toiminta voi tuntua etäännyttävältä ja hankalalta, tekoäly arkipäiväistyy koko ajan ja valmiita tekoäly ratkaisuja on jo olemassa. Kuvio 5 näyttää tiivistetysti asiat, jotka tutkimuksen perusteella tulee huomioida, kun tekoälyavusteista testausta lähdetään kehittämään.



KUVIO 5 Tekoälyn käyttöönotto testauksen tehostamiseen

8 LOPPUSANAT

Kappaleessa käydään läpi tutkimuksen yhteenveto. Yhteenvedossa tarkastellaan, miten tutkimuskysymyksiin vastattiin, tutkimuksen rajoitukset sekä jatkotutkimusaiheet. Kirjallisuuskatsauksessa nousi esiin, että tekoälyä on hyödynnetty ohjelmistotestaukseen, joten ohjelmistojen laadunvarmistus rajattiin ohjelmistotestaukseen. Kirjallisuuskatsauksen avulla luotiin aiheeseen yleiskatsaus. Tämä sisälsi tekoälyn tieteenalana ja käytännössä, ohjelmistotestauksen sekä mitä tekoälyn hyödyntäminen ohjelmistotestauksessa tällä hetkellä on.

8.1 Vastaukset tutkimuskysymyksiin

Viimeaikaiset tutkimukset osoittavat, että tekoäly ohjelmistotestauksessa on laajasti tunnistettu ohjelmistoalan trendi, joka kiinnostaa ja puhututtaa ohjelmistoalaa. Tutkimuksen avulla haluttiin selvittää, onko tekoäly jalkautunut ohjelmistotestaukseen käytännön tasolla. Vastauksia tutkimukseen lähdettiin hakemaan seuraavien tutkimuskysymysten avulla:

- Miten tekoälyä hyödynnetään ohjelmistotestauksessa IT-organisaatioissa Suomessa?
- Millä tavalla tekoäly on muuttanut testausta organisaatiossa?

Tutkimusmenetelmäksi valikoitui kvalitatiivinen monitapaustutkimus. Aineistonkeruu toteutettiin kvalitatiivisella teemahaastattelumenetelmällä. Tutkimukseen valikoitui 8 haastateltavaa seitsemästä organisaatiosta. Haastateltavat olivat pääasiassa ohjelmistotestauksen parissa uransa tehneitä, mutta myös datatieteilijä ja ohjelmistokehittäjä sisältyi haastateltaviin. Tutkimuksen tulokset osoittavat, että tekoälyä hyödynnetään käyttöliittymätestaukseen, testiautomaatioon sekä testitapausten priorisointiin. Ratkaisut ovat olleet kokeiluja ja yksittäisiä, räätälöityjä ratkaisuja asiakkaille. Tekoäly ole vielä jalkautunut osaksi ohjelmistotestausta. Haastateltavat kertoivat, että kokeiluista on saatu lupaavia

tuloksia. Tarvitaan lisää asiakastapauksia ja dataa, jotta kehitystä voidaan viedä eteenpäin.

Tekoälyn avulla on saatu eliminoitua testiautomaation manuaalista ylläpitoa. Yksi haastateltava kertoi asiakkaalle räätälöidystä tekoälyratkaisusta. Ratkaisun avulla toisistaan eroavien fyysisten laitteiden manuaalinen testaus oli tekoälyn avulla poistunut kokonaan. Tekoäly osasi jaotella oikeat testit oikeille laitteille, eikä ihmisen tarvinnut manuaalisesti tehdä jaottelua. Toinen esimerkki oli, että tekoälyn avulla saatiin luotua automaattinen testi tuotannosta raportoidun manuaalisen virhekuvauksen perusteella. Testi toistaa virheen, jolloin juurisyihin päästään käsiksi. Tekoäly ei ole muuttanut vielä testausta, vaan testausta tehdään testiautomaatiota ja tutkivaa testausta hyödyntäen. Tekoälyavusteinen testaus on vielä marginaali-ilmiö suomalaisissa IT-organisaatioissa. Tekoälyä kohtaan on kuitenkin kiinnostusta ja kehitystyö on lähtenyt käyntiin.

8.2 Tutkimuksen rajoitukset ja jatkotutkimusaiheet

Tutkimuksen aineistonkeruu toteutettiin teemahaastattelun avulla. Teemahaastattelun avulla aineistoa saadaan kerättyä joustavasti ja tilannetta voi myötäillä haastateltavan mukaan. (Hirsjärvi, ym., 2003, s.194–197; Hirsjärvi & Hurme, 2008, s. 48.) Tekijällä oli käytännön kokemusta ohjelmistokehityksestä ja yksikkötestauksesta, mutta ei ohjelmistotestauksesta ja testiautomaatiosta. Tällä saattoi olla vaikutusta haastattelukysymysten laadinnassa ohjelmistotestauksesta.

Kun haastateltavilta kysyttiin, millaista tekoälyä testausratkaisuissa oli käytetty, monikaan haastateltava ei osannut vastata tähän, koska eivät välttämättä olleet mukana koko projektin ajan. Haastateltavien kuvailemien tekoälyratkaisujen visualisointi olisi voinut antaa tekijälle paremman käsityksen, miten kehitetty ratkaisu toimii. Tekijä ei itse tätä kysynyt haastateltavilta. Yksi haastateltavista lähetti haastattelun päätteeksi linkin opinnäytetyöhönsä sekä kuvakaappauksia ja videoita kehitetystä RPA ratkaisusta, mikä havainnollisti ratkaisua laajemmin, kuin pelkkä keskustelu.

Tutkimuksen luotettavuutta pyrittiin varmistamaan siten, että haastateltaville lähetettiin kysymykset etukäteen ja he saivat perehtyä teemoihin ennen haastattelua. Kaikki haastattelut nauhoitettiin kahdella laitteella, jolloin aineiston analyysissa päästiin palaamaan tarvittaessa keskusteluun. Tässä tutkimuksessa löydettiin tapoja, joilla tekoälyä on hyödynnetty ohjelmistotestaukseen. Sen vaikutuksia testaukseen laajemmalla skaalalla ei voitu vielä kuitenkaan arvioida, koska kehitystyö on vielä alkutekijöissään. Hyödyllinen jatkotutkimusaihe olisi tutkia kehitetyn tekoälyratkaisun vaikutuksia testaukseen oikeassa asiakasprojektissa.

LÄHTEET

Amannejad, Y., Garousi, V., Irving, R., & Sahaf, Z. (2014). A search-based approach for cost-effective software test automation decision support and an industrial case study. *In 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops* (pp. 302-311). IEEE.

Applitools. (2021). Applitools Eyes. Haettu 29.11.2021 osoitteesta https://applitools.com/products-eyes/

Bellapu, A. (29.1.2021). *How Can Artificial Intelligence Aid In Software Testing.* Analytics Insight. [Blogikirjoitus]. Haettu 18.11.2021 osoitteesta https://www.analyticsinsight.net/how-can-artificial-intelligence-aid-in-software-testing/

Bisht, S. (2013). *Robot framework test automation*. Packt Publishing.

Bruderer, H. (2016). The birth of artificial intelligence: first conference on artificial intelligence in paris in 1951?. In IFIP *International Conference on the History of Computing* (pp. 181-185). Springer, Cham.

Brunette, E. S., Flemmer, R. C., & Flemmer, C. L. (2009). A review of artificial intelligence. *In 2009 4th International Conference on Autonomous Robots and Agents* (pp. 385-392). Ieee.

Cruzes, D. S., & Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. *In 2011 international symposium on empirical software engineering and measurement* (pp. 275-284). IEEE.

Eisenhardt, K. M. (1989). Building theories from case study research. *Academy of management review*, 14(4), 532-550.

Elements of AI. (2021). Elements of AI. Online course. *Reaktor & University of Helsinki.* Haettu 13.10.2021 osoitteesta https://www.elementsofai.com/

Eriksson, P. & Koistinen, K. (2014). *Monenlainen tapaustutkimus*. Helsinki: Kuluttajatutkimuskeskus.

Everett, G. D., & McLeod, R. (2007). *Software testing: Testing across the entire software development life cycle*. IEEE Press..

Fogg, E. (2021). The Impact of AI on Software Quality Assurance. Embedded Computing Design. [blogikirjoitus]. Haettu osoitteesta https://www.embeddedcomputing.com/technology/ai-machine-learning/ai-dev-tools-frameworks/the-impact-of-ai-on-software-quality-assurance

Functionize. (2022). Future-proof your Testing with Big Data, Machine Learning, & Computer Vision. Haettu 15.11.2021 osoitteesta https://www.functionize.com/

Gao, J., Tao, C., Jie, D., & Lu, S. (2019, April). What is AI software testing? and why. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)* (pp. 27-2709). IEEE.

Garousi, V., Felderer, M., & Mäntylä, M. V. (2019). Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology,* 106, 101-121.

Gartner Inc. (2020). Top Priorities for IT: Leadership Vision for 2021. Emerging trends, expected challenges and next steps for CIOs and IT leaders in 2021. Haettu 10.10.2021 osoitteesta https://emtemp.gcom.cloud/ngw/globalassets/en/publications/documents/top-priorities-for-it-leadership-vision-for-2021-ebook.pdf

Goericke, S. (2020). *The future of software quality assurance* (p. 257). Springer Nature.

Graham, D., & Fewster, M. (2012). *Experiences of test automation: case studies of software test automation.* Addison-Wesley Professional.

Hambling, B., Morgan, P., Samaroo, A., Thompson, G., & Williams, P. (2010). *Software Testing.* British Informatics Society Limited.

Hirsjärvi, S., Remes, P. & Sajavaara, P. (2003). *Tutki ja kirjoita.* (10. uud. painos). Helsinki: Tammi.

Hooda, I., & Chhillar, R. S. (2015). Software test process, testing types and techniques. *International Journal of Computer Applications*, 111(13).

Hourani, H., Hammad, A., & Lafi, M. (2019). The Impact of Artificial Intelligence on Software Testing. *2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)* (pp. 565-570). IEEE.

International Organization for Standardization. (2017). Systems and software engineering – vocabulary (ISO/IEC/IEEE 24765:2017). Haettu 14.10.2021 osoitteesta https://ieeexplore-ieee-org.ezproxy.jyu.fi/stamp/stamp.jsp?tp=&arnumber=8016712

JavaTpoint. (2021.) RPA Vs. Test Automation. Haettu 29.12.2021 osoitteesta https://www.javatpoint.com/rpa-vs-test-automation

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. Science, 349(6245), 255-260.

Kananen, J. (2008). *Kvali – Kvalitatiivisen tutkimuksen teoria ja käytänteet.* Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kaplan, J. (2016). *Artificial intelligence: What everyone needs to knowR*. Oxford University Press

Kayid, A. (2020). The role of Artificial Intelligence in future technology. *Department of Computer Science.* The German University in Cairo. 10.13140/RG.2.2.12799.23201.

King, T. M., Arbon, J., Santiago, D., Adamo, D., Chin, W., & Shanmugam, R. (2019). AI for testing today and tomorrow: industry perspectives. 2019 *IEEE International Conference On Artificial Intelligence Testing (AITest)* (pp. 81-88). IEEE.

Kirdey, S., Cureton, K., Rick, S., Ramanathan, S., Shukla, M. (2019). Lerner — using RL agents for test case scheduling. The Netflix Technology Blog. Medium. [blogikirjoitus]. Haettu 17.10.2021 osoitteesta https://netflixtechblog.com/lerner-using-rl-agents-for-test-case-scheduling-3e0686211198

Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR),* 52(6), 1-35.

Leung, H. K., & White, L. (1990). A study of integration testing and software regression at the integration level. In Proceedings. *Conference on Software Maintenance 1990* (pp. 290-301). IEEE.

Machalica, M., Samylkin, A., Porth, M. & Chandra, S. (2018). Predictive test selection: A more efficient way to ensure reliability of code changes. Facebook Engineering. [blogikirjoitus]. Haettu 29.11.2021 osoitteesta https://engineering.fb.com/2018/11/21/developer-tools/predictive-test-selection/

Malviya, R. (2021). Leveraging AI in Quality Assurance. Infosys. [Blogikirjoitus]. Haettu 29.11.2021 osoitteesta https://www.infosys.com/insights/ai-automation/quality-assurance.html

Mili, A. & Tchier, F. (2015). *Software testing: Concepts and operations.* John Wiley & Sons, Inc.

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., & Yu, B. (2019). Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592.*

Myers, M. D., & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. *Information and organization,* 17(1), 2-26.

Ning, S., & Yan, M. (2010). Discussion on research and development of artificial intelligence. *2010 IEEE International Conference on Advanced Management Science(ICAMS 2010),* 110–112

Pati, S. (3.8.2021). The Difference Between Artificial Intelligence and Machine Learning. Analytics Insight. [Blogikirjoitus]. Haettu 18.11.2021 osoitteesta https://www.analyticsinsight.net/the-difference-between-artificial-intelligence-and-machine-learning/

Pittet, S. (2021). The different types of software testing. Atlassian. [Blogikirjoitus]. Haettu 10.10.2021 osoitteesta

https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

QA Madness. (2021). The Future Is Here: Top Software Testing Trends 2020. [Blogikirjoitus]. Haettu 10.10.2021 osoitteesta https://www.qamadness.com/the-future-is-here-top-software-testing-trends-2020/

Quadri, S. M. K., & Farooq, S. U. (2010). Software testing–goals, principles, and limitations. *International Journal of Computer Applications,* 6(9), 1.

Ramchand, S., Shaikh, S., & Alam, I. (2021). Role of Artificial Intelligence in Software Quality Assurance. In *Proceedings of SAI Intelligent Systems Conference* (pp. 125-136). Springer, Cham.

Rana, I., Goswami, P., & Maheshwari, H. (2019). A review of tools and techniques used in software testing. *Int. J. Emerg. Technol. Innovative Res,* 6(4), 262-266.

Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence,* 1(5), 206-215.

Rudin, C., & Radin, J. (2019). Why are we using black box models in AI when we don't need to? A lesson from an explainable AI competition. *Harvard Data Science Review,* 1(2), 10-1162.

Russell, S. J., Chang, M., Devlin, J., Dragan, A., Forsyth, D., Goodfellow, I., . . . Wooldridge, M. (2022). *Artificial intelligence: A modern* approach (Fourth edition. Global edition.). Pearson.

Santiago, D., King, T. M., & Clarke, P. (2018). AI-Driven test generation: machines learning from human testers. In *Proceedings of the 36th Pacific NW Software Quality Conference* (pp. 1-14).

Shi, Z. (2011). *Advanced Artificial Intelligence.* World Scientific.

Singh, Y. (2011). *Software testing.* Cambridge University Press.

Software Testing Fundamentals. (2020). All code is guilty until proven innocent. Haettu 13.11.2021 osoitteesta https://softwaretestingfundamentals.com/

Sommerville, I. (2016*). Software engineering* (Tenth edition, global edition.). Pearson.

Spieker, H., Gotlieb, A., Marijan, D., & Mossige, M. (2017, July). Reinforcement learning for automatic test case prioritization and selection in continuous integration. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 12-22).

Stresnjak, S., & Hocenski, Z. (2011). Usage of robot framework in automation of functional test regression. *In Proc. 6th Int. Conf. Softw. Eng. Adv.(ICSEA)* (pp. 30-34).

Suomen virallinen tilasto (SVT). (2021). Tietotekniikan käyttö yrityksissä. Helsinki: Tilastokeskus. Haettu 12.10.2021 osoitteesta http://www.stat.fi/til/icte/2021/icte_2021_2021-12-03_tie_001_fi.html

Test.ai. (2022). AI-First-or-Fail. Haettu 20.11.2021 osoitteesta https://test.ai/

TestingXpert. (2022). AI in Software Testing – Benefits, Approaches, Tools to Look in 2022. [Blogikirjoitus]. Haettu 12.10.2021 osoitteesta https://www.testingxperts.com/blog/AI-in-Software-Testing#What%20is%20AI-based%20Testing?

Tieturi Oy. (16.12.2021). Miten AI auttaa ohjelmistotestausta? – Tieturi webinaari, [video]. Youtube. Haettu 14.12.2021 osoitteesta https://www.youtube.com/watch?v=CN1fc2_HoGA&ab_channel=TieturiOy

Tricentis (2017). Test Automation vs. Automated Testing: The Difference Matters. [Blogikirjoitus]. Haettu 12.10.2021 osoitteesta https://www.tricentis.com/blog/test-automation-vs-automated-testing-the-difference-matters/

Tuominen, H., Neittaanmäki, P., Niinimäki, E., Pölönen, I., Rautiainen, I., Äyrämö, S., ... & Äyrämö, S. M. (2019). Tekoälyn perusteita ja sovelluksia-Jyväskylän yliopisto: Informaatioteknologian tiedekunta. Kurssiaineisto.

Virtanen, J. (2021). Nämä ovat 3 kuuminta it-trendiä juuri nyt. Tivi. [Blogikirjoitus] Haettu 2.10.2021 osoitteesta https://www.tivi.fi/uutiset/nama-ovat-3-kuuminta-it-trendia-juuri-nyt/e8555ac3-9b7a-4e28-8429-35897504da68

Volk, T. (2021). Disrupting the Economics of Software Testing Through AI. Enterprise Management Associates (EMA). Haettu 10.10.2021 osoitteesta https://ewig5qf9cgn.exactdn.com/wp-content/uploads/2021/09/EMA-Applitools-DisruptingTestingEconomics-092121.pdf

Wagner, S. k. (2013). Software Product Quality Control. Springer Berlin Heidelberg.

Wood., J. (2021). What's the difference between Test Automation and RPA? DEV Community. [Blogikirjoitus]. Haettu 20.11.2021 osoitteesta https://dev.to/woodjessica/what-s-the-difference-between-test-automation-and-rpa-21h6

LIITE 1 HAASTATTELURUNKO

Haastattelun aloitus ja taustatiedot

- Esittäytyminen
- Tutkimuksen esittely, tarkoitus ja näkökulmat
- Lupa haastattelun nauhoittamiseen

Haastattelun teemat ja kysymykset

1. Haastateltavan taustat

a. Nykyinen rooli yrityksessä ja mitä työ sisältää.

b. Montako vuotta olet työskennellyt ohjelmistotestauksen parissa?

2. Tekoäly

a. Miten määrittelisit tekoälyn? Mitä tekoäly mielestäsi on?

3. Testaus

a. Voitko kuvailla jonkun teidän ohjelmistoprojektin testausprosessia?

b. Mitkä osat testausprosessissa tehdään testaajan toimesta manuaalisesti?

c. Mitkä osat testausprosessissa on automatisoitu?

4. Tekoäly testauksessa

a. Minkälaisia haasteita testauksessa on, joihin toivotaan ratkaisua tekoälyn avulla?

b. Minkälaista tekoälyä testauksessa on hyödynnetty tai suunnitellaan hyödynnettävän? Esim. RPA, NLP, konenäkö, koneoppiminen yms.

c. Mihin testauksen vaiheisiin tekoälyä on hyödynnetty tai suunnitellaan hyödynnettävän?

d. Millä tavalla tekoälyn odotetaan vaikuttavan testaukseen?

e. Minkälaisia haasteita on tunnistettu tekoälyn hyödyntämisessä testaukseen?

f. Jatkokehityssuunnitelmat tekoälyn hyödyntämiseen testauksessa.