Aalto-yliopisto

Perustieteiden korkeakoulu

Master's Programme in Computer, Communication and Information Sciences

Emmi Mannonen

Vähäkoodisen ohjelmistokehityksen vaikutus tekoälyn selitettävyyteen ja ymmärrettävyyteen

Diplomityö

Helsinki, 1. huhtikuuta 2022

Valvoja: Professori Marjo Kauppinen

Ohjaaja: Diplomi-insinööri Manu Setälä

Aalto-yliopisto
Aalto-universitetet
Aalto University

Tekijä Emmi Mannonen

Työn nimi Vähäkoodisen ohjelmistokehityksen vaikutus tekoälyn selitettävyyteen ja ymmärrettävyyteen

Koulutusohjelma Master's Programme in Computer, Communication and Information Sciences

Pääaine Software and Service Engineering

Vastuuopettaja/valvoja Professori Marjo Kauppinen

Työn ohjaaja(t) Diplomi-insinööri Manu Setälä

Yhteistyötaho Solita Oy

Päivämäärä 01.04.2022 **Sivumäärä** 90+1 **Kieli** Suomi

Tiivistelmä

Vähäkoodinen ohjelmistokehitys on aiheena ajankohtainen. Vähäkoodisella ohjelmistokehityksellä voidaan kehittää ohjelmistoja graafisten komponenttien avulla siten, että perinteistä koodia tarvitaan joko vähän tai ei lainkaan. Tämän diplomityön tavoitteena on tutkia, miten vähäkoodinen ohjelmistokehitys vaikuttaa tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta.

Tämän diplomityön empiirisen tutkimuksen lähestymistavaksi valittiin konstruktiivinen tutkimusmenetelmä ja se toteutettiin tapaustutkimuksen ja puolistrukturoitujen haastattelujen avulla. Tutkimuksessa toteutettiin olemassa olevaan rekrytointipeliin lisäosa, jossa pelaajalle annetaan suosituksia kohdeyrityksen avoimista työpaikoista. Lisäosa toteutettiin OutSystems-kehitysalustalla, joka on tekoälyä hyödyntävä vähäkoodinen ohjelmistokehitysalusta. Lisäosan demonstroinnin yhteydessä haastatteluun valituille henkilöille esiteltiin OutSystems-kehitysalustan avulla tuotetut graafiset prosessimallit.

Tutkimustuloksista selvisi, että vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit lisäävät tekoälyä hyödyntävän sovelluksen ymmärrettävyyttä ja selitettävyyttä. Haastateltavat olivat yksimielisiä siitä, että graafiset prosessikaaviot auttoivat ymmärtämään työpaikkojen suosittelu -osion toimintalogiikkaa. Tekoälyn selitettävyys parani, kun tilaaja pystyi tunnistamaan ohjelmistossa käytettävän datan ja ymmärtämään, mistä data oli peräisin ja miten sitä käsiteltiin.

Avainsanat Vähäkoodinen ohjelmistokehitys, Vähäkoodinen ohjelmistokehitysalusta, Low-code, Tekoäly, Selitettävä tekoäly

Aalto-yliopisto
Aalto-universitetet
Aalto University

Author Emmi Mannonen

Title of thesis The impact of low-code on the explainability and understandability of AI

Programme Master's Programme in Computer, Communication and Information Sciences

Major Software and Service Engineering

Thesis supervisor Prof. Marjo Kauppinen

Thesis advisor(s) M.Sc Manu Setälä

Collaborative partner Solita Oy

Date 01.04.2022 **Number of pages** 90+1 **Language** English

Abstract

Low-code software development will play an increasingly important role in future software development. Low-code software development allows software to be developed using graphical components with little or no need for traditional coding. The aim of this thesis is to investigate how low-code software development affects the understandability and explainability of artificial intelligence (AI) for the client.

Design science was chosen as the empirical research method for the thesis. The research was conducted through a case study and semi-structured interviews. In the case study, an extension to an existing recruitment game was developed. In the developed extension, the player is given recommendations on job vacancies in the target company. The extension was implemented on the OutSystems which is low-code development platform. During the presentation of the extension, the interviewees were shown graphical process models produced on the OutSystems development platform.

The results of the empirical study showed that graphical models provided by a low-code development platform support the understandability and explainability of AI-based system. The interviewees agreed that the graphical process diagrams helped to understand the functional logic of the job recommendation section. The explainability of the AI-based system improved when the client was able to identify the data used in the software, understand where the data came from and how it was processed.

Keywords Low-code software development, Low-code platform, Artificial intelligence, Explainable AI

Aalto-yliopisto
Aalto-universitetet
Aalto University

Sisällys

Esipuhe 8
Lyhenteet 9
1 Johdanto 10
1.1 Tausta ja motivaatio 10
1.2 Tutkimusongelma ja kysymykset 11
1.3 Tutkielman rakenne 12
2 Tutkimusmenetelmät 14
2.1 Kirjallisuuskatsaus 14
2.2 Empiirinen tutkimus 17
2.2.1 Tapaustutkimuksen kohdeyritys ja -projekti 17
2.2.2 Tutkimusprosessi 18
2.2.3 Datan keräys ja analysointi 21
3 Kirjallisuuskatsaus 24
3.1 Vähäkoodinen ohjelmistokehitys 24
3.1.1 Mitä on vähäkoodinen ohjelmistokehitys 24
3.1.2 Vähäkoodiset ohjelmistokehitysalustat 25
3.1.3 Vähäkoodisen ohjelmistokehityksen hyödyt 28
3.1.4 Vähäkoodisen ohjelmistokehityksen riskit 30
3.1.5 Vähäkoodisen ohjelmistokehityksen vaiheet 31
3.1.6 Vähäkoodisten menetelmien käyttökohteet 34
3.2 Tekoälyn ymmärrettävyys ja selitettävyys 34
3.2.1 Mitä on tekoälyn ymmärrettävyys ja selitettävyys 34
3.2.2 EU:n uudet säädökset luotettavalle tekoälylle 36
3.2.3 Selittävä tekoäly 38
3.2.4 Tekoälyn ja ihmisen yhteistyö 41
3.2.5 Tekoälyn päätöksenteon puutteet 43
3.2.6 Miten parantaa tekoälyn ymmärrettävyyttä ja selitettävyyttä 44
3.3 Yhteenveto kirjallisuuskatsauksen tuloksista 46
4 Empiirinen tutkimus 49
4.1 Case study: työpaikkojen suosittelu -sovellus 49
4.1.1 Kuvaus pelin olemassa olevista toiminnallisuuksista 50

4.1.2 Sovelluksen suunnittelu ... 53
4.1.3 Sovelluksen toteutus ... 55
4.1.4 Sovelluksen käyttöönotto ... 68
4.2 Haastattelut ... 68
4.2.1 Vähäkoodinen ohjelmistokehitys ... 68
4.2.2 Tekoäly ... 70
4.2.3 Visuaalisuuden vaikutus tekoälyn ymmärrettävyyteen ja selitettävyyteen ... 72
4.3 Yhteenveto empiirisen tutkimuksen tuloksista ... 76
5 Pohdinta ... 78
5.1 RQ1: Vähäkoodisella ohjelmistokehityksellä tuotettu sovellus ... 78
5.2 RQ2: Vähäkoodisen kehitysalustan graafisten mallien vaikutus tekoälyn selitettävyyteen ja ymmärrettävyyteen ... 79
5.3 Tutkimuksen rajoitteet ... 81
6 Johtopäätökset ... 83
Lähteet ... 86
LIITTEET ... 91

Esipuhe

Aloittaessani opinnot Aalto-yliopistossa vuonna 2017, minulle oli selvää jo ensimmäisen jakson jälkeen, etten valmistuisi odotetussa viidessä vuodessa. Vielä varmempaa siitä tuli ajan edetessä; tuli upeita ystäviä, vapaaehtoistöitä, unohtumattomia ja unohdettuja tapahtumia, pääsin kokemaan vaihdon Singaporessa ja aloittamaan työurani työskentelemällä kaksi vuotta kokopäiväisesti opintojen ohella. Jos vielä vuosi sitten minulle olisi sanottu, että valmistun vuoden päästä määräajassa, olisin naurahtanut ja sanonut, että ”katellaan”. Tässä sitä kuitenkin ollaan, kirjoittamassa esipuhetta diplomityöhöni ja viiden vuoden täyttymiseenkin on vielä 5 kuukautta.

Tämä ei todellakaan olisi ollut mahdollista ilman ympärilläni olevia ihmisiä. Ensimmäiseksi haluan kiittää Solitaa luottamuksesta, että pääsin tekemään diplomityöni juuri teille ja siitä, että vielä diplomityön jälkeenkin haluatte jatkaa yhteistyötä vakituisen työsuhteen muodossa. Kiitos AIGA-hankkeelle, että sain tuoda low-code näkökulmaa tärkeään aihepiiriinne. Kiitos Manu Setälä ja Antti Rannisto mahtavasta sparrauksesta, neuvoista ja ohjauksesta. Kiitos Marjo Kauppinen; olit erittäin hyvä ”valvoja”, vaikket sitä termiä haluaisikaan käyttää. Annoit palautteesi erittäin rakentavasti ja opin palautteistasi ja neuvoistasi paljon. Arvostan erityisen paljon tapaasi painottaa sitä, että diplomityöni on ja oli minun työni ja minä itse päätin sen sisällöstä.

Kiitos etenkin täystuhot, sekä Paitsio ja Skipoli, että teitte opiskeluvuosistani unohtumattomat ja opiskelusta huumorilla maustettua. Kiitos lihapäät sydänkohtauksista, mutta myös parhaista nauruista. Kiitos Aapo kaikesta tuesta ja muhun uskomisesta, etenkin niinä aikoina, kun en edes itse uskonut itseeni. Viimeisenä iso kiitos koko perheelleni, oikeastaan kaikesta. Etenkin kiitos sulle äiti, että oot jaksanut jo ala-asteelta asti opiskella mun kanssa, vaikka mua useasti kiinnosti enemmän seisoa päällä, kun opetella englantia. Tuskin musta maisteria ois tullu ilman sun tukea, ainakin tämän diplomityön kanssa hakkaisin vieläkin päätäni seinään.

Helsingissä 1.4.2022
Emmi Mannonen

Lyhenteet

AI	Tekoäly
LCDP	Vähäkoodinen ohjelmistokehitysalusta: graafinen käyttöliittymä, jossa sovellus rakennetaan pudottamalla ja raahaamalla valmiita elementtejä
ML	Koneoppiminen
XAI	Selittävä tekoäly

1 Johdanto

Tässä luvussa tutustutaan tutkimuksen syihin ja motivaatioon, esitetään tutkimusongelma ja tutkimuskysymykset sekä käydään läpi diplomityön rakenne ja laajuus.

1.1 Tausta ja motivaatio

Yhä useampi ohjelmisto käyttää hyödykseen tekoälyalgoritmeja, jotka avustavat ihmisiä työssä ja päätöksenteossa. EU tuoreessa asetusehdotuksessaan (COM/2021/206 final) pyrkii vahvistamaan tekoälyn oikeusvarmuutta ja ihmisten luottamusta tekoälyyn. EU:n vaatimukset liittyvät muun muassa tekoälyn selitettävyyteen ja läpinäkyvyyteen. Käyttäjän tulee pystyä ymmärtämään, milloin hän on vuorovaikutuksessa tekoälyn kanssa. EU ottaa kantaa ohjelmistohankkeiden lisäksi myös tekoälyä hyödyntävien järjestelmien elinkaareen. Tekoälyä sisältävä ohjelmisto voi sisältää uhkia, joita ei ollut olemassa ohjelmiston kehityksen alussa, vaan niitä on muodostunut ohjelmistopäivitysten ja koneoppimisen myötä vasta tuotantokäytön aikana [44].

Tekoälyjärjestelmien kehittäminen on perinteisesti vaatinut paljon rahaa ja korkeasti koulutettuja tietoteknisiä osaajia. Nykypäivänä tekoälyä hyödyntäviä ohjelmistoja ei rakenneta pelkästään perinteisillä ohjelmistokehitysmenetelmillä, vaan rinnalle on tullut vähäkoodinen ohjelmistokehitys (low-code). Monien asiantuntijoiden arvio on, että vähäkoodisilla ohjelmistokehitysmenetelmillä järjestelmien kehittäminen on nopeampaa ja siksi niistä on ennustettu tulevan merkittävä tapa kehittää ohjelmistoja [40].

Tekoälyn kehittämiseen on perinteisesti vaadittu korkeaa matemaattista ja tietoteknistä osaamista. Vähäkoodisessa ohjelmistokehityksessä tämä korkean tason osaamisvaatimus on vähäisempi, sillä tekoälyä sisältäviä ohjelmistoja voidaan tuottaa jo ohjelmistokehittämisen peruslogiikkaa ymmärtämällä. Vähäkoodisen ohjelmistokehittämisen ydin on graafinen käyttöliittymä, jota kehittäjät käyttävät tiedon ja työnkulkujen mallintamiseen vedä-ja-pudota -menetelmillä [34]. Koska ohjelmistojen kehittäminen vähäkoodisilla kehitysalustoilla ei vaadi syvää ohjelmointiosaamista, yhä useammalla on valmiuksia rakentaa tekoälyä sisältäviä ohjelmistoja. Tähän voi piillä myös riskejä, sillä ohjelmistokehityksen laatu ja järjestelmien väliset integraatiot voivat kärsiä uusien ei-teknisten osaajien myötä. Muun muassa sovellusarkkitehtuuri nousee tässä merkittävään rooliin [27].

OutSystems teetti vuonna 2019 kyselytutkimuksen, jonka mukaan lähes viidennes suomalaisista ajattelee koodaamisen olevan osa tulevaisuuden työnkuvaa [45]. Kansalaiskehittämisellä tarkoitetaan sitä, että ohjelmiston loppukäyttäjä kehittää liiketoimintaprosesseja ja työtään tukevia ohjelmistoja ilman syvempää ohjelmointiosaamista [45]. Kansalaiskehittämisessä nähdään paljon potentiaalia. Se voi poistaa raja-aitoja ammattiryhmien välillä ja se voi tuoda esiin piilossa olevaa osaamista [27]. Ohjelmointiosaamista tarvitaan edelleen vaativien ja monimutkaisten järjestelmien rakentamisessa, johon vähäkoodinen ohjelmistokehittäminen ei ainakaan vielä taivu.

On tärkeää, että vähäkoodisilla ohjelmistokehitysmenetelmillä pystytään vastaamaan tekoälyn ymmärrettävyys ja selitettävyys -vaatimuksiin. Selitettävyys tarkoittaa, että ihminen voi ymmärtää mihin tekoälyjärjestelmän tekemät päätökset perustuvat [14]. Ihmisen luottamusta tekoälyjärjestelmään lisää se, jos hän ymmärtää tekoälyn toimintaa ja tunnistaa, kuka on vastuussa sen tavasta toimia [14]. Kun tekoälyjärjestelmä on toteutettu vähäkoodisella kehitysalustalla graafisten komponenttien avulla, tekoäly voi avautua helpommin myös ohjelmiston tilaajalle. Tällöin on mahdollista, että tekoälyratkaisuista pystytään yhä enenevässä määrin keskustelemaan, niiden toimintaa pystytään kritisoimaan ja niiden tekemiin päätöksiin vaaditaan perusteluita.

Valitsin vähäkoodisen ohjelmistokehityksen ja tekoälyn päätöksentekoon liittyvän ymmärrettävyyden ja selitettävyyden diplomityöni aiheeksi siksi, että asia kiinnostaa minua ja aihe on juuri nyt ajankohtainen. Uskon, että vähäkoodisella ohjelmistokehityksellä tulee olemaan iso rooli tulevaisuuden ohjelmistokehityksessä sekä Suomessa että maailmalla. Samaan uskoo myös Gartner, joka ennustaa, että vuoteen 2024 mennessä yli 65 prosenttia sovelluksista toteutetaan vähäkoodisilla kehitysalustoilla [34].

1.2 Tutkimusongelma ja kysymykset

Diplomityö perustuu kahden vahvan megatrendin yhteisvaikutukseen. Gartner nostaa hypekäyrällään erittäin korkealle sekä AI:n että AI Governancen [17]. Toinen ilmiö, jonka uskotaan nousevan megatrendiksi, on vähäkoodinen ohjelmistokehitys. Syy vähäkoodisen ohjelmistokehityksen nousuun on se, että vähäkoodisen ohjelmistokehityksen uskotaan olevan nopeampaa, helpompaa ja kustannustehokkaampaa kuin perinteinen proseduraalista ohjelmointia sisältävä ohjelmistokehitys [40]. Näihin edellä mainittuihin hyötyihin uskovat monet muutkin toimijat sekä Suomessa että ulkomailla. HS Visio kirjoitti aiheesta 16.12.2021 artikkelissaan Kooditon tulevaisuus [22]. Siinäkin uskotaan, että vähäkoodiset tavat toteuttaa ohjelmistoja ovat tulleet jäädäkseen.

Tässä diplomityössä tarkastellaan sitä, miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit avaavat tekoälyn logiikkaa ja päätöksentekoa ohjelmiston tilaajalle. On tärkeää, että ohjelmiston tilaaja pystyisi ymmärtämään tekoälyn toimintaa ja sitä, mihin tekoälyn tekemät päätökset perustuvat.

Diplomityössä selvitetään myös EU:n uusia linjauksia, jotka liittyvät tekoälyn selittävyyteen ja läpinäkyvyyteen. Tutkimuksessa pyritään tekemään näkyväksi se, mitkä asiat vaikuttavat vähäkoodisella kehitysalustalla tuotetun tekoälyn ymmärrettävyyteen sekä miten nämä asiat eroavat perinteisistä tavoista tuottaa ohjelmistoja.

Diplomityön pääongelmana tutkitaan sitä, miten vähäkoodinen ohjelmistokehitys vaikuttaa tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta.

Tutkimusongelmaa lähestytään seuraavilla tutkimuskysymyksillä:

RQ1: Miten vähäkoodisella ohjelmistokehityksellä luodaan tekoälyä hyödyntäviä ohjelmistoja?

RQ2: Miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta?

Tekoälyn tekemät päätökset perustuvat tiedon loogiseen päättelyyn, kun taas ihmisen tekemissä päätöksissä on mukana intuitiota ja päättelevää älyä [26]. Tekoälyn objektiivinen lähestymistapa voi joskus olla jopa ristiriidassa ihmisen tekemän intuitiivisen päätöksen kanssa. Ihmisen on helpompi hyväksyä tekoälyn tekemät päätökset, jos niiden perusteet ovat helposti nähtävissä ja ymmärrettävissä. Tässä diplomityössä pyritään antamaan myös suosituksia siitä, millä keinoilla vähäkoodisella kehitysalustalla tuotetun ohjelmiston selitettävyyttä ja ymmärrettävyyttä voitaisiin parantaa. Lisäksi pyritään antamaan suosituksia siitä, millaiseen tekoälykehitykseen vähäkoodista ohjelmistokehitystä kannattaisi käyttää.

1.3 Tutkielman rakenne

Tutkimuskysymyksiin etsitään vastauksia sekä kirjallisuuskatsauksen että empiirisen tutkimuksen avulla. Diplomityö koostuu johdannon lisäksi viidestä pääluvusta. Diplomityö etenee ilmiötä käsittelevästä kirjallisuuskatsauksesta empiirisen tutkimuksen analysointiin asti.

Luvussa 2 esitellään tutkimusmenetelmät ja kuvataan empiirisen tutkimuksen tutkimusprosessi. Kyseisessä luvussa on lueteltu kirjallisuuskatsauksessa käytetyt tieteelliset artikkelit ja muut julkaisut. Luvussa 3 käydään läpi kirjallisuuskatsauksen tulokset, joissa kirjallisuutta on tutkittu tutkimuskysymysten näkökulmasta. Kirjallisuuskatsaus jakautuu kahteen osaan. Ensimmäinen osa käsittelee vähäkoodista ohjelmistokehitystä ja toinen osa tekoälyn ymmärrettävyyttä ja selitettävyyttä. Luvun lopussa on yhteenveto.

Luvussa 4 kuvataan empiirisen tutkimuksen tuloksia. Luku aloitetaan esittelemällä tutkimuksen kohteena olevan rekrytointipelin olemassa olevat toiminnallisuudet ja sen jälkeen kerrotaan työpaikkojen suosittelu -osiosta, joka on toteutettu vähäkoodisella kehitysalustalla tätä tutkimusta varten. Luvussa 5 esitetään yhteenveto kuhunkin tutkimuskysymykseen liittyvistä tärkeimmistä tuloksista, arvioidaan miten hyvin tutkimuskysymyksiin pystyttiin vastaamaan ja kerrotaan tutkimuksen rajoitteet. Diplomityö päättyy luvun 6 johtopäätöksiin, jossa esitetään lyhyet vastaukset tutkimuskysymyksiin ja ehdotetaan aiheita jatkotutkimuksille.

2 Tutkimusmenetelmät

2.1 Kirjallisuuskatsaus

Diplomityön kirjallisuuskatsaus koostuu kahdesta osiosta: vähäkoodisesta ohjelmistokehityksestä (low-code) sekä tekoälyyn liittyvästä ymmärrettävyydestä ja selitettävyydestä. Tutkimuksessa on käytetty OutSystems-kehitysalustaa, joka sisältää tekoälyavustajan. Tutkimuksessa yhdistyvät molemmat kirjallisuuskatsauksen osiot ja siksi juuri ne on valittu kirjallisuuskatsauksen teemoiksi.

Kirjallisuuslähteet kerättiin hyödyntäen Google Scholar -hakupalvelua sekä Scopus-tietokantaa. Vähäkoodisiin kehitysalustoihin löytyy jo jonkin verran tieteellistä kirjallisuutta. Aihetta on täydennetty myös internetistä löytyneillä tiedoilla, joita ovat blogit, uutiset ja IT-toimittajien toimittamat materiaalit. Netistä löytyneisiin tietoihin on pitänyt suhtautua varauksella ja samaa asiaa on pitänyt tarkastella monen lähteen näkökulmasta, jotta tekstin sisältöön on voinut luottaa. Jotkut lauseet olen tarkoituksella kirjoittanut tyyliin "...voi aiheuttaa, ...uskotaan olevan", koska alkuperäisestä lähdemateriaalista ei ole löytynyt tutkimusnäyttöä. Näin olen tehnyt muun muassa vähäkoodisen ohjelmistokehityksen hyödyistä kirjoittaessani, sillä esimerkiksi kansalaiskehittäjien todellisista onnistumisista ei ole löytynyt tutkimustietoa.

Selittävään tekoälyyn löytyy paljon tieteellistä lähdemateriaalia. Näihin materiaaleihin on voinut luottaa enemmän, sillä niihin löytyy enemmän tutkimusnäyttöä. Myös EU on julkaissut paljon tekoälyyn liittyvää materiaalia. Tutkimusta tehdessä olisi ollut mahdollista porautua vaikka kuinka syvälle EU:n tekoälyyn liittyviin selvityksiin ja lainsäädäntöön. Tässä on kuitenkin rajauduttu tutkimaan vain tuoretta tekoälyasetusehdotusta, jotta diplomityö pysyisi suunnitellussa laajuudessaan.

Kaikki tämän diplomityön aiheeseen liittyvä lähdemateriaali on varsin tuoretta. Varsinkin vähäkoodiseen ohjelmistokehitykseen liittyvä materiaali on tuotettu vuosien 2019-2021 aikana. Hyvänä puolena näen sen, ettei lähdemateriaali sisällä vanhentunutta tietoa. Taulukossa 1 esitellään kirjallisuuskatsauksessa käytetyt hakusanat sekä niiden avulla löydetyt tieteelliset artikkelit ja muu lähdemateriaali.

Taulukko 1: kirjallisuuskatsauksen hakusanat ja niiden perusteella löydetyt tieteelliset artikkelit ja muu tärkeä lähdemateriaali

Hakusanat	**Tieteelliset artikkelit**	**Muu tärkeä lähdemateriaali**
Low-code / low code Low-code development Low-code platform Low-code AI	Alonso et al., Towards a Polyglot Data Access Layer for a Low-Code Application Development Platform, 2020 Jacinto et al., Test Mocks for Low-Code Applications built with OutSystems, 2020 Sahay et al., Supporting the understanding and comparison of low-code development platforms, 2020 Sanchis et al., Low-Code as Enabler of Digital Transformation in Manufacturing Industry, 2020 Tisi et al., Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms, 2019 Waszkowski, Low-code platform for automating business processes in manufacturing, 2019 Woo, The Rise of No/Low Code Software Development-No Experience Needed?, 2020	Digital Illustrated, 5 myyttiä low-code kehittämisestä, blogi 2021 eCraft, Mitä on low-code ja kuka sitä tarvitsee? blogi 2021 Forward Forever, Low-code -alustojen mahdollisuudet PK-yrityksille, esitys 2021 Lyytinen, IT:ssä herätys - kansalaiskehittäjät tulevat!, Efima blogi 2020 OutSystems, Outsystems platform features. Www-sivusto Retool, What is low code? A comprehensive guide 2021 Solita, Nopeampaa ohjelmistokehitystä automaation avulla, blogi 2021

Hakusanat	Tieteelliset artikkelit
AI XAI Understandable AI Explainable AI	Adadi et al., Peeking inside the black-box: A survey on explainable artificial intelligence (XAI), 2021 Ashoori et al., In AI We Trust? Factors That Influence Trustworthiness of AI-infused Decision-Making Processes, 2019 Floridi et al., AI4People—An Ethical Framework for a Good AI Society: Opportunities, Risks, Principles, and Recommendations, 2018 Hagras, Toward Human-Understandable, Explainable AI, 2018 Jarrahi, Artificial Intelligence and the Future of Work: Human-AI Symbiosis in Organizational Decision Making, 2018 Miller, Explanation in artificial intelligence: Insights from the social sciences, 2019 Rossi et al. A definition of AI: Main Capabilities and Disciplines, 2019 Samek et al, Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models, 2017 Schneider et al., Personalized explanation for machine learning, 2019

EU tekoälyasetus:

- Euroopan komissio, Luotettavaa tekoälyä koskevat eettiset ohjeet, 2019
- Työ- ja elinkeinoministeriö, Valtioneuvoston kirjelmä eduskunnalle komission ehdotuksesta Euroopan parlamentin ja neuvoston asetukseksi tekoälyn harmonisoiduksi sääntelyksi, 2021

2.2 Empiirinen tutkimus

Tämän kappaleen tarkoituksena on antaa yleiskatsaus empiirisen tutkimuksen toteuttamisesta. Valitsin konstruktiivisen tutkimuksen empiirisen tutkimuksen tutkimusmenetelmäksi. Empiirinen tutkimus toteutettiin tapaustutkimuksen sekä puolistrukturoitujen haastattelujen avulla.

2.2.1 Tapaustutkimuksen kohdeyritys ja -projekti

Tässä kappaleessa esitellään tapaustutkimuksen ympäristö. Kappaleessa esitellään kohdeyritys ja kohdeprojekti. Lisäksi esitellään AIGA-hanke, jonka osana tämä diplomityö on. Näiden esittelyjen avulla voidaan ymmärtää paremmin ongelma-aluetta ja luoda viitekehys seuraaville luvuille.

Tämä diplomityö tehdään osaksi AIGA-hanketta (Artificial Intelligence Governance and Auditing). AIGA-hankkeen ja täten myös tämän diplomityön rahoittajana toimii Business Finland. Turun kauppakorkeakoulun koordinoimassa AIGA-hankkeessa keskitytään tekoälyn tuottamien päätösten luotettavuuden, läpinäkyvyyden ja ymmärrettävyyden vahvistamiseen. Hankkeessa kehitetään tekoälyn hallintamalleja ja -mekanismeja, joilla pyritään varmistamaan tekoälyn eettisyys jo ohjelmiston kehitysvaiheessa. Turun yliopistosta hankkeeseen osallistuvat Turun kauppakorkeakoulu, tulevaisuuden teknologioiden laitos sekä oikeustieteellinen tiedekunta. Yhteistyötahoina ovat lisäksi Verohallinto ja Helsingin yliopisto sekä joukko IT-alan yrityksiä. Lisätietoa AIGA-hankkeesta löytyy https://ai-governance.eu/.

Case Company

Tutkimuksen kohdeyritys on Solita. Vuonna 1996 perustettu Solita on teknologia-, data- ja designyritys, joka kehittää IT-palveluita, sähköistä asiointia sekä tiedolla johtamisen ratkaisuja yrityksille ja julkishallinnon organisaatioille. Solita työllistää yli tuhat strategisen konsultoinnin, palvelumuotoilun, ohjelmistokehityksen, analytiikan, pilvipalvelujen ja integraation osaajaa Suomessa, Ruotsissa, Tanskassa, Virossa, Belgiassa ja Saksassa. Solita ja OutSystems aloittivat low-code-yhteistyön Suomessa ja Ruotsissa vuonna 2019.

Case Project

Tapaustutkimuksena toteutettiin OutSystems-kehitysalustan avulla tekoälyä hyödyntävä työpaikkojen suosittelu -osio. Kyseinen osio toteutettiin olemassa olevan rekrytointipelin lisäosaksi. Työpaikkojen suosittelu -osio antaa pelaajalle suosituksia siitä, mihin kohdeyrityksen avoimeen työpaikkaan

hänen kannattaisi hakea. Pelissä käyttäjä suorittaa eri ohjelmointikielillä ohjelmointitehtäviä, jotka vievät häntä pelissä eteenpäin. Pelaaja voi milloin tahansa lopettaa pelaamisen tai vaihtaa "huonetta" eli ohjelmointikieltä. Lopetettuaan pelin pelaajalta kysytään nimeä, sähköpostia, työkokemusta sekä opiskelutaustaa. Näiden taustatietojen ja pelattujen pisteiden perusteella tekoäly suosittelee pelaajalle avoimia työpaikkoja kohdeyritykseltä, joka tässä tutkimuksessa on Solita. Pelaaja voi myös olla luovuttamatta tietojaan, jolloin peli näyttää pelaajalle suosituksia työpaikoista vain pelattujen pisteiden perusteella. Tarkemmat tiedot pelin toiminnallisuuksista löytyvät kohdasta 4.1. Case study: työpaikkojen suosittelu -sovellus.

Sovelluksen tilaajana tässä tapaustutkimuksessa pidetään Solitan HR-yksikköä. HR-yksikkö syöttää sovellukseen avoimet työtehtävät ja osaamisvaatimukset, joita tietyssä avoimessa työpaikassa tarvitaan. Sovelluksen tilaaja määritellään siksi, että diplomityön tutkimuskysymyksessä tekoälyn ymmärrettävyyttä ja selitettävyyttä tarkastellaan myös tilaajan näkökulmasta.

Tapaustutkimuksen tarkoituksena oli tutkia, miten tekoälyä luodaan vähäkoodisen ohjelmistokehityksen avulla. Tapaustutkimusta analysoidessa käytin teorialähtöistä analyysimenetelmää, jossa pyrin todistamaan oikeaksi kirjallisuuskatsauksen tuottaman tuloksen. Tavoitteenani oli osoittaa, miten tekoälyn luomisprosessi etenee ja millä menetelmillä sitä voidaan muokata. Kun voidaan nähdä ja todistaa, miten tekoälyä luodaan, on mahdollista tunnistaa keinoja, jotka voivat vaikuttaa tekoälyn ymmärrettävyyteen.

2.2.2 Tutkimusprosessi

Valitsin empiirisen tutkimuksen tutkimusmenetelmäksi konstruktiivisen tutkimuksen (design science), koska tutkimuksen tavoitteena oli suunnitella ja toteuttaa konkreettinen lopputulos eli sovellus. Konstruktiivisen tutkimuksen lähtökohta on organisaation todellinen ongelma, johon konstruktiivisessa tutkimuksessa kehitetään ratkaisu [24].

Diplomityön pääongelmana tutkittiin sitä, miten vähäkoodinen ohjelmistokehitys vaikuttaa tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta. Tutkimusprosessi aloitettiin kirjallisuuskatsauksella, jossa keskityttiin vähäkoodiseen ohjelmistokehitykseen sekä tekoälyn ymmärrettävyyteen ja selitettävyyteen liittyvään kirjallisuuteen. Tutkimuksen toinen vaihe suoritettiin tapaustutkimuksena kohdeyrityksessä, jossa artefakti (sovellus) toteutettiin ja sen toimivuutta arvioitiin kohdeyrityksen kontekstissa haastattelujen avulla. Artefakti oli vähäkoodisella kehitysalustalla toteutettu yksinkertainen tekoälyä hyödyntävä sovellus, jonka tuottajana toimin minä, eli tutkija. Näin pääsin aktiivisesti osallistumaan

ongelmanratkaisuprosessiin. Ratkaisu kehitettiin yhdessä kohdeyrityksen ja tilaajan (HR-osaston) kanssa, ja tilaaja kuului myös haastattelujen kohderyhmään.

Konstruktiivisen tutkimus on tutkimuslähestymistapa, jonka tavoitteena on ratkaista merkityksellisiä ja käytännöllisiä haasteita iteratiivisen ja systemaattisen ongelmanratkaisuprosessin avulla [24]. Konstruktiivisen tutkimuksen ytimessä on sellaisen artefaktin tuottaminen, jolla vastataan yrityksen kannalta merkitykselliseen ongelmaan. Artefaktin tulee tuottaa merkityksellinen ratkaisu yrityksen vielä ratkaisemattomaan liiketoimintaongelmaan [50]. Lisäksi konstruktiivisessa tutkimuksessa korostetaan, että tutkija osallistuu aktiivisesti ongelmanratkaisuprosessiin ja että ratkaisuja kehitetään yhteistyössä organisaation kanssa sen sijaan, että vain havainnoitaisiin tapahtumia tai testattaisiin teorioita havaintojen perusteella [24].

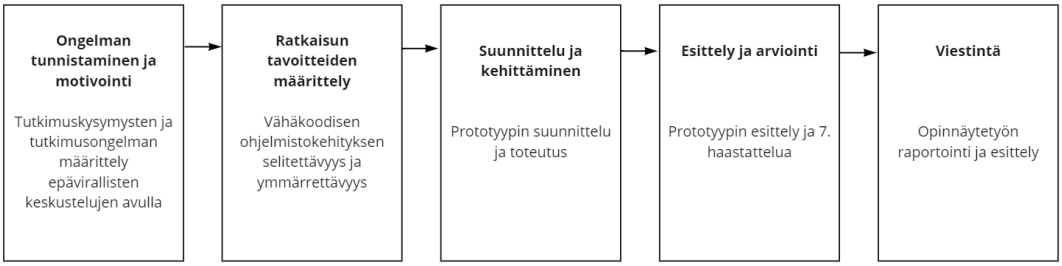
Konstruktiivisen tutkimuksen tutkimusprosessi sisältää useita aktiviteetteja, joita Peffersin et al. [50] mukaan ovat

1. ongelman tunnistaminen
2. ratkaisun tavoitteiden määrittely
3. suunnittelu ja kehittäminen
4. esittely
5. arviointi
6. viestintä

Tutkimuksen ensimmäisenä aktiviteettina on ongelman tunnistaminen, jonka avulla kehitetään artefakti. Kun ongelma on tunnistettu ja todettu tärkeäksi, se motivoi tutkijaa etsimään siihen ratkaisua. Seuraavaksi määritellään ratkaisun tavoitteet ja toteutuskelpoisuus. Tavoitteet voivat olla joko kvantitatiivisia tai kvalitatiivisia. Peffers et al. muistuttavat, että tavoitteet tulee johtaa rationaalisesti ongelman määrittelystä [50].

Kolmantena aktiviteettina on ratkaisun suunnittelu ja kehittäminen, eli artefaktin luominen. Tähän tehtävään kuuluvat artefaktin toiminnallisuuden ja arkkitehtuurin määrittäminen sekä varsinaisen artefaktin luominen [50]. Neljäs aktiviteetti on artefaktin esitteleminen eli demoaminen. Demonstraatio voi tarkoittaa esimerkiksi ratkaisun käyttöä kokeilussa, simuloinnissa tai tapaustutkimuksessa. Seuraavassa aktiviteetissa arvioidaan, miten hyvin toteutettu ratkaisu tukee tunnistetun ongelman ratkaisua [50]. Arviointivaiheessa verrataan ratkaisun tavoitteita demonstraatiossa saatuihin tuloksiin, Peffer et al. täsmentävät. Arviointi voi tapahtua eri muodossa riippuen artefaktin luonteesta. Tämän aktiviteetin lopussa tutkija voi halutessaan palata artefaktin uudelleen suunnitteluun. Tällainen iterointi ei ole aina mahdollista. Tutkimusprosessin viimeinen aktiviteetti on ratkaisusta viestiminen.

Kuvassa 1 on esitetty tutkimusprosessi, jota käytettiin tässä tutkimuksessa. Prosessissa on kuvattuna kaikki design science tutkimusmenetelmän kohdat, mutta demonstraatio ja arviointi on yhdistetty samaan sarakkeeseen, sillä ne toteutettiin samassa aktiviteetissa.



Kuva 1. Tässä tutkimuksessa käytetty tutkimusprosessi

Empiirinen tutkimus käynnistyi ongelman tunnistamisella. Tämän diplomityön ongelmaksi valikoitui aihe, joka liittyy vähäkoodisen ohjelmistokehittämisen kasvuun Solitalla ja AIGA-hankkeessa tutkittavan tekoälyn ymmärrettävyyteen ja selitettävyyteen. Tutkimusongelma määriteltiin perehtymällä tarkemmin aiheeseen ja keskustelemalla tutkimuksen ohjaajan ja valvojan kanssa. Tutkimusongelmaa päätettiin lähestyä kahdella tutkimuskysymyksellä, jotka koettiin tutkimuksen kannalta merkityksellisiksi. Tutkimuskysymykset olivat seuraavat: “miten vähäkoodisella ohjelmistokehityksellä luodaan tekoälyä hyödyntäviä ohjelmistoja” ja “miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta”.

Kun tutkimusongelma oli tunnistettu, määriteltiin ratkaisun tavoitteet ja sen toteutuskelpoisuus. Konstruktiivisen tutkimusmenetelmän mukaisesti tavoitteet johdettiin rationaalisesti ongelman määrittelystä. Tavoitteet määriteltiin yhteistyössä tilaajan kanssa.

Empiirisen tutkimuksen kolmantena aktiviteettina oli artefaktin suunnittelu ja kehittäminen. Artifaktina suunniteltiin ja toteutettiin tekoälyä hyödyntävä työpaikkojen suosittelu -lisäosa. Lisäosa kehitettiin tavoitteiden mukaisesti OutSystemsilla, joka on vähäkoodinen kehitysalusta. Artifaktin suunnittelu ja arkkitehtuuri laadittiin yhteistyössä Solitan asiantuntijoiden kanssa.

Kun sovellus oli valmis, sitä demonstroitiin haastatteluun valitetuille henkilöille. Tässä tutkimuksessa artefaktin toimintaa demonstroitiin haastatteluiden yhteydessä. Haastateltaville demonstroitiin toteutetun työpaikkojen suosittelu -lisäosan toiminta, sekä esitettiin kyseisen lisäosan taustalla toimivat OutSystems-kehitysalustan avulla tuotetut graafiset prosessimallit.

Demonstraation lisäksi haastatteluihin yhdistettiin myös konstruktiivisen tutkimusmenetelmän viides aktiviteetti eli arviointi. Demonstraation jälkeen haastateltavilta kysyttiin tutkimuksen tavoitteisiin liittyviä kysymyksiä ja pyrittiin muodostamaan kattavaa kuvaa siitä, kuinka artefakti täytti nämä tavoitteet.

Tutkimusprosessin viimeinen aktiviteetti oli viestiminen eli tämän diplomityön kirjoittaminen, sekä diplomityön ohjaajalle ja valvojalle pidetty esitelmä. Diplomityön ratkaisuista pidettiin esitelmä myös AIGA-hankkeelle diplomityön kirjoittamisen loppupuolella.

2.2.3 Datan keräys ja analysointi

Tutkimuksessa käytettiin puolistrukturoituja haastatteluja, jotta voitiin selvittää, miten vähäkoodisessa ohjelmistokehityksessä käytetyt menetelmät vaikuttavat tekoälyn selitettävyyteen ja ymmärrettävyyteen. Valitsin puolistrukturoidun haastattelun, koska se antaa haastatteluun liikkumavaraa strukturoituun haastatteluun nähden. Puolistrukturoidussa haastattelussa esitetyt kysymykset laadittiin etukäteen. Tämä helpottaa sitä, että haastattelu pysyy aiheessa. Kysymysten järjestystä ja vastausmuotoja voidaan tarvittaessa muuttaa, samoin osa kysymyksistä voidaan jättää pois haastattelun edetessä. Haastattelun lopputuloksen kannalta on tärkeää, että haastattelija olisi mahdollisimman puolueeton, vaikka haastattelija voikin kysymyksillään jossain määrin ohjata haastateltavia. Mahdolliset kommentit ja lisäkysymykset on esitettävä neutraalisti ja vältettävä antamasta sellaisia viestejä, jotka voivat ohjata haastateltavan vastauksia [37]. Haastattelu on hyvä nauhoittaa, koska niiden ansiosta haastattelut voidaan raportoida tarkemmin [37].

Haastattelujen avulla pyrittiin vastaamaan toiseen tutkimuskysymykseen eli “miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta”. Haastattelukysymykset laadittiin tukemaan tätä tutkimuskysymystä. Haastattelu alkoi johdannolla haastattelun tavoitteista ja periaatteista, sekä haastateltavien taustatietojen selvittämisellä. Tämän jälkeen siirryttiin yleisiin vähäkoodiseen ohjelmistokehitykseen ja tekoälyyn liittyviin kysymyksiin, joiden avulla pyrittiin selvittämään haastateltavan ajatukset ja mielipiteet käsiteltävästä asiasta.

Ennen yleisiä kysymyksiä haastateltavalle ei annettu mitään tietoa käsiteltävistä aiheista. Vasta yleisten kysymysten jälkeen siirryttiin rekrytointipelin demoamiseen ja sen toteutuksen esittelyyn. Demonstraatiossa näytettiin, kuinka varsinainen peli ja työpaikkojen suosittelu -osio toimivat sekä näytettiin miten jälkimmäinen kehitettiin OutSystemsillä. Esittelyssä näytettiin

muun muassa OutSystemsin avulla tuotetut graafiset prosessimallit. Demonstraation jälkeen haastateltavalle esitettiin demoon liittyviä kysymyksiä. Lopuksi tehtiin yhteenveto tutkimuksesta ja keskusteltiin vapaasti aiheesta haastateltavan kanssa. Diplomityön liitteenä 1 olevat haastattelukysymykset jaettiin johdannon lisäksi kolmeen ryhmään:

- Yleistä vähäkoodisesta ohjelmistokehityksestä ja tekoälystä
- Rekrytointipeli ja OutSystems-toteutus
- Yhteenveto tutkimuksesta ja nykyisistä löydöksistä.

Haastattelut toteutettiin helmikuussa 2022 Microsoft Teamsin välityksellä. Kaikki haastattelut tehtiin suomeksi, koska haastatteluun osallistujat ovat suomenkielisiä. Suurin osa haastatteluista kesti 45 minuuttia ja muutama haastattelu kesti reilun tunnin. Aikaa mahdollisiin ylityksiin oli varattu, eikä kiirettä ollut. Jokainen haastattelu nauhoitettiin. Nauhoituksen aikana tehtiin myös lyhyitä muistiinpanoja, jotta haluttuihin kysymyksiin ja vastauksiin päästiin palaamaan nopeammin puhtaaksikirjoittamisen yhteydessä. Haastatteluaineistoa nauhoitettiin yhteensä kuusi tuntia ja kaksitoista minuuttia.

Haastattelukutsuja lähetettiin Solitan henkilökunnalle pääasiassa niille HR-osaston henkilöille, jotka olivat rekrytointipelin tilaajina. Kutsuja lähetettiin myös vähäkoodisen ohjelmistokehityksen parissa työskenteleville, perinteisen ohjelmistokehityksen edustajille ja projektijohtajalle. Haastatteluihin ilmoittautui seitsemän henkilöä ja kaikki heistä pääsi osallistumaan. Kolme haastateltavista oli HR:n henkilöstöä, jotka vastaavat etenkin rekrytoinnista. Kaksi haastateltavista oli vähäkoodisen ohjelmistokehityksen edustajia, joista toinen työskentelee OutSystemsin parissa ja toinen Microsoftin Power Platformin parissa. Haastatteluun pääsivät myös yksi perinteisen ohjelmoinnin edustaja ja yksi projektipäällikkö/asiakasvastaava. Neljä haastateltavista oli naisia ja kolme miehiä. Haastateltavat ovat iältään 24-54 -vuotiaita. Taulukossa 2 on yhteenveto haastateltavista henkilöistä. Haastateltujen roolit ovat jaettu low-code kehittäjä, ohjelmistokehittäjä, HR ja projektipäällikkö. Tarkkoja nimikkeitä ei ole käytetty henkilöiden yksityisyyden suojaamiseksi.

Puhtaaksikirjoittamisen jälkeen haastatteluiden sisältö analysoitiin luokittelemalla vastaukset. Luokittelussa pyrittiin löytämään aihekokonaisuuksia sekä toistuvia linjauksia ja mielipiteitä. Puhtaaksi kirjoitetusta tekstistä ryhmiteltiin pääaiheet esimerkiksi “vähäkoodinen ohjelmistokehitys”, “ymmärrettävyys” ja “tekoäly”, jotka jaettiin myöhemmin pienempiin ryhmiin esimerkiksi “vähäkoodisen ohjelmistokehityksen vaikutukset”, “tekoälyn lähestyttävyys” ja “graafisten komponenttien ymmärrettävyys”. Lopuksi ryhmät jaoteltiin vielä positiivisiin ja negatiivisiin esimerkiksi “positiiviset vähäkoodisen ohjelmistokehityksen vaikutukset” ja “negatiiviset vähäkoodisen ohjelmistokehityksen vaikutukset”.

Taulukko 2: Yhteenveto haastatteluun osallistuvista henkilöistä.

Rooli	**Vähäkoodisen ohjelmistokehityksen osaaminen**	**Tekninen osaaminen**	**Tilaajan edustaja tässä tutkimuksessa**
HR	Vähän	Kyllä, koulutuksen kautta	x
HR	Vähän	Ei	x
HR	Ei yhtään	Kyllä, koulutuksen ja työn kautta	x
Low-code kehittäjä	Paljon	Kyllä, koulutuksen ja työn kautta	
Low-code kehittäjä	Paljon	Kyllä, harrastuneisuuden ja työn kautta	
Ohjelmistokehittäjä	Vähän	Kyllä, koulutuksen ja työn kautta	
Projektipäällikkö	Ei yhtään	Kyllä, koulutuksen ja työn kautta	

3 Kirjallisuuskatsaus

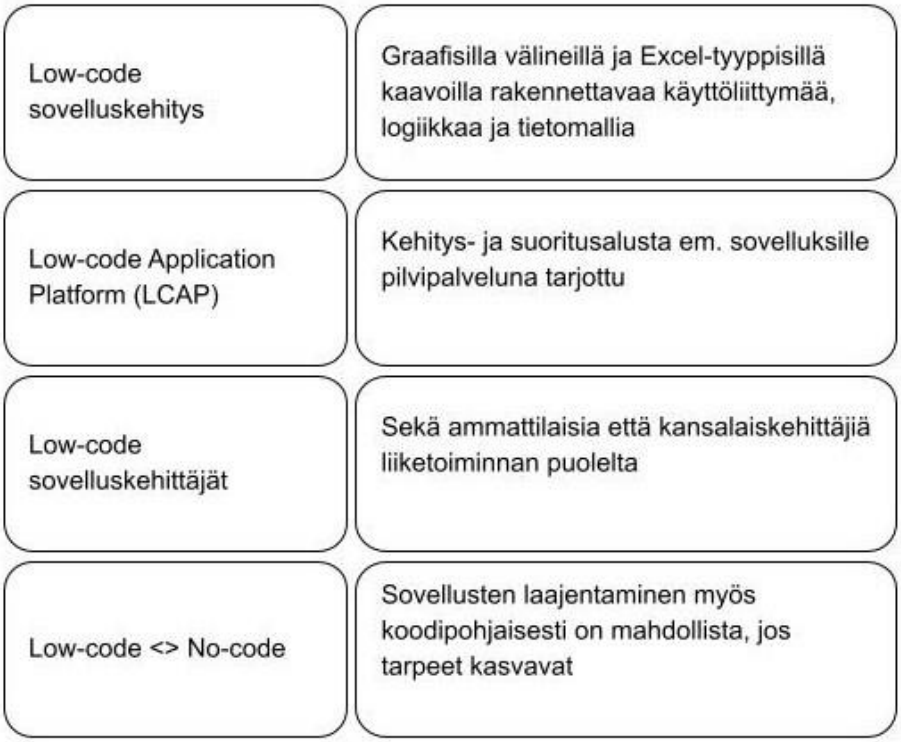
Tässä luvussa tehdään katsaus olemassa olevaan kirjallisuuteen diplomityön kannalta merkityksellisistä aiheista. Aluksi kerrotaan, mitä vähäkoodisella ohjelmistokehityksellä tarkoitetaan ja miten vähäkoodinen ohjelmistokehitystyö käytännössä etenee. Lisäksi käydään läpi millaisia hyötyjä ja riskejä sen käytössä on tunnistettu. Seuraavaksi siirrytään tarkastelemaan tekoälyn ymmärrettävyyttä ja selitettävyyttä käsitteinä: mitä niillä tarkoitetaan ja miksi ne ovat tärkeitä ominaisuuksia tekoälyjärjestelmien kehittämisessä. Tässä luvussa pohditaan myös sitä, voidaanko tekoälyn ymmärrettävyyttä ja selitettävyyttä parantaa jollain keinoilla. Luvun lopussa on yhteenveto kirjallisuuskatsauksen tuloksista.

3.1 Vähäkoodinen ohjelmistokehitys

Ensimmäinen tutkimuskysymys on, miten vähäkoodisella ohjelmistokehityksellä luodaan tekoälyä hyödyntäviä sovelluksia. Seuraavissa kappaleissa pyrin selvittämään, mitä vähäkoodinen ohjelmistokehitys (low-code) tarkoittaa sekä millaisia hyötyjä ja riskejä siinä on tunnistettu. Lisäksi kuvataan, mitä tarkoitetaan vähäkoodisilla ohjelmistokehitysalustoilla.

3.1.1 Mitä on vähäkoodinen ohjelmistokehitys

Ohjelmistojen kehittäminen vähäkoodisilla kehitysalustoilla perustuu siihen, että kehittäjät toteuttavat ohjelmistoja pääasiallisesti raahaamalla ja pudottamalla valmiita komponentteja [2]. Työn tulos saadaan näkyviin nopeasti. Vähäkoodisen ohjelmistokehityksen ideana on, että suurin osa työstä voidaan tehdä käyttämättä aikaa perinteisen koodin kirjoittamiseen. Kun valitut komponentit ovat paikoillaan, vähäkoodinen kehitysalusta tuottaa koodin taustalla [49]. Vähäkoodiseen ohjelmistokehitykseen kuuluu oleellisesti kansalaiskehittäminen. Ohjelmiston kehittäjää, joilla ei ole ohjelmointitaustaa, kutsutaan kansalaiskehittäjäksi [43]. Kuvassa 2 Forward Forever on tiivistänyt vähäkoodisen ohjelmistokehittämisen peruskäsitteet [16].



Kuva 2: Vähäkoodisen ohjelmistokehittämisen peruskäsitteet [16].

3.1.2 Vähäkoodiset ohjelmistokehitysalustat

Yritysten ja myös julkisen sektorin tulee reagoida nopeasti muuttuviin teknologiavaatimuksiin. Tästä syystä markkinoille on tullut kehitystyökaluja, joilla pyritään lisäämään ohjelmistokehittämisen automatisointia [40]. Jo pitkään monet tutkimusprojektit ovat yrittäneet mahdollistaa nopeaa ohjelmistokehitystä, jossa sovelluksia pystyttäisiin kehittämään ilman perinteistä proseduraalista ohjelmointia [43]. 1990-luvulla otettiin käyttöön CASE-teknologian työkaluja, joiden tavoitteena oli automatisoida ohjelmistojen kehittämistä. CASE-työkalujen käyttö jäi kuitenkin odotuksia vähäisemmäksi, vaikka tarve automatisoinnin lisäämiseen tunnistettiin [40]. Tarve on yhä olemassa ja nykypäivänä automatisointia tukevat kehitysalustat ovat alkaneet vallata markkinoita yhä enemmän. Nämä kehitysalustat tunnetaan nimellä vähäkoodiset ohjelmistokehitysalustat (low-code development platforms, LCDP) [43, 38].

Valmiiden visuaalisten elementtien avulla vähäkoodisia kehitysalustoja pystytään käyttämään tehokkaasti myös ilman ohjelmointikokemusta [43]. Vähäkoodiset kehitysalustat tarjoavat mahdollisuuden kirjoittaa proseduraalista koodia, mikäli ohjelmistoon halutaan luoda monipuolisempia ominaisuuksia kuin mitä alustan valmiilla komponenteilla olisi mahdollista.

Vähäkoodiset kehitysalustat eivät toimi ainoastaan kehitysalustoina, vaan ne tarjoavat myös tehokkaan ympäristön sovelluksen koko elinkaarelle, mukaan lukien kehitys, toteutus, käyttöönotto ja ylläpito [43]. Vähäkoodisia kehitysalustoja käytetään erityisesti web- ja mobiilisovellusten kehittämiseen ja niitä tarjotaan Platform-as-a-Service -mallilla [38]. Platform-as-a-Servicella tarkoitetaan sitä, kun palveluntuottaja tarjoaa sovellusalustan palveluna ohjelmistokehityksen käyttöön [7]. Pilvipohjaiset vähäkoodiset kehitysalustat tarjoavat kehitys- ja suoritusalustan, jossa merkittävässä osassa ovat monipuoliset graafiset käyttöliittymät ja visuaaliset kaaviot. Waszkowski on tutkinut Aurea BPM l -kehitysalustan käyttöä teollisuuden liiketoiminnan prosessien automatisoinnissa. Hän uskoo, että vähäkoodisten kehitysalustojen avulla on mahdollista luoda toimivia ohjelmistoja ja ottaa ne käyttöön vähäisellä tai olemattomalla määrällä proseduraalista koodia [48].

OutSystems-ohjelmistokehitysalusta

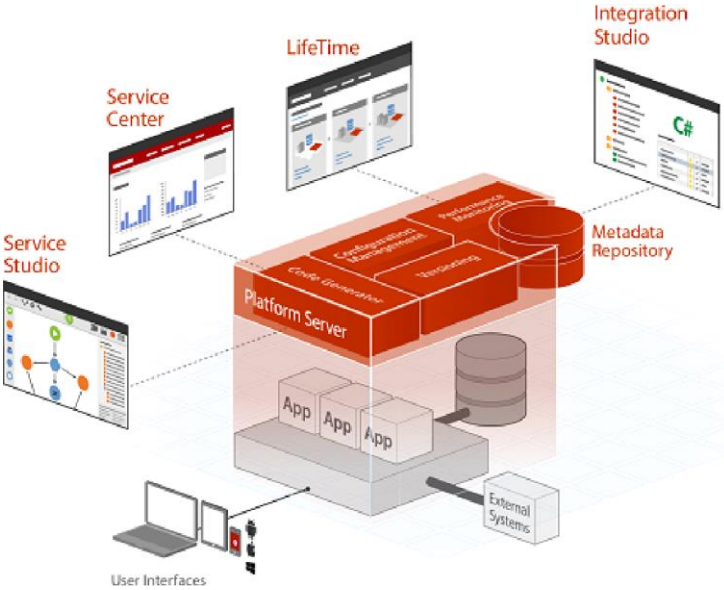
Markkinoilla on useita vähäkoodisia kehitysalustoja, joiden perusajatus on sama eli visuaalinen mallintaminen koodin käsinkirjoittamisen sijaan. Tässä tutkimuksessa käytettiin OutSystems-ohjelmistokehitysalustaa. Gartner vertaili vuonna 2021 vähäkoodisia ohjelmistokehitysalustoja ja tässä vertailussa OutSystems Platform sijoittui kärkipäähän [18]. Kuvassa 3 esitetään Gartnerin tekemän vertailun tulos.

Kuva 3: Gartnerin vuonna 2021 tekemä vähäkoodisten kehitysalustojen vertailu [18].

OutSystems sisältää valmiita komponentteja ja -työkaluja, joiden avulla kehittäjä voi rakentaa sovelluksia [25]. OutSystemsin tekoälyavustaja auttaa ohjelmistokehityksessä kehittämisen eri työvaiheissa. Tekoälyavustaja ehdottaa kehittäjille oikeita työkaluja ja malleja sekä pyrkii ehkäisemään virheiden syntyä [32]. Tekoäly tukee ohjelmistokehitystä, mutta se sisältää myös työkaluja tekoälysovellusten kehittämiseen [32]. Tällaisia työkaluja ovat muun muassa automatisoidun päätöksenteon tuki ja chatbot-toiminnot. Automatisoitu päätöksenteon tuki koostuu koneoppimismalleista, jotka perustuvat sovellukseen aiemmin tallennetuista päätöksistä. Näitä malleja voidaan kouluttaa käytön aikana, jolloin sovelluksen tekemät päätökset tarkentuvat oppimisen myötä [32].

OutSystemsissä sovellusten käyttöliittymä, toimintalogiikka ja tietomallit luodaan visuaalisesti raahaamalla ja pudottamalla suunnittelunäkymään valmiita elementtejä. Kehitysalusta muodostaa koodin taustalla. Ohjelmistoon voidaan tarvittaessa lisätä JavaScript-, Java-, C#-, SQL-, CSS- tai HTML-koodia käsin [21]. Valmis sovellus voidaan julkaista yhdellä napin painalluksella joko pilvessä tai yrityksen omassa ympäristössä.

OutSystems tarjoaa tehokkaan ympäristön sovelluksen koko elinkaarelle, mukaan lukien kehitys, toteutus, käyttöönotto ja ylläpito [25]. OutSystemsiä käytetään liiketoiminnan järjestelmien uudistamisessa, mobiili- ja web-sovellusten kehityksessä sekä asiakaskokemuksen parantamiseen esimerkiksi erilaisten itsepalvelusovellusten kehityksessä.



Kuva 4: OutSystems kehitysalustan arkkitehtuuri [25].

OutSystemsin arkkitehtuuri koostuu viidestä osasta, joita ovat Platform Server, Integration Studio, LifeTime, Service Studio ja Service Center [25, 32]. Arkkitehtuuri on kuvattu kuvassa 4. Platform Server on kokoelma yhteentoimivia komponentteja, joita käytetään sovelluksen kehittämiseen, tuottamiseen, hallitsemiseen, optimoimiseen ja käyttöönottoon. Integration Studio on OutSystemsin osa, jossa rakennetaan integraatiot kolmansien osapuolten järjestelmiin ja palveluihin. Näitä integraatioita voidaan käyttää niitä uudelleen muissakin OutSystems-sovelluksissa. LifeTime on konsoli, joka mahdollistaa kehitys-, testi- ja tuotantoympäristöjen keskitetyn hallinnan [3].

OutSystemsilla toteutetut sovellukset kehitetään Service Studiossa. Service Studio on jaettu välilehtiin, joita ovat Käyttöliittymä, Logiikka, Data ja Prosessit [25]. Välilehtien nimi kertoo niiden käyttötarkoituksen. Käyttöliittymä-osiossa kehittäjä luo ohjelmiston käyttöliittymät raahaamalla ja pudottamalla valitut komponentit oikeisiin paikkoihin. Logiikka-osiossa luodaan ohjelmiston toiminnot. Data-osiossa kehittäjä määrittelee tietokantamallin ja Prosessit-osiossa liiketoimintaprosessit ja työnkulut.

3.1.3 Vähäkoodisen ohjelmistokehityksen hyödyt

eCraft on nostanut blogissaan vähäkoodisen ohjelmistokehityksen hyödyiksi kehittäminen nopeuden, kustannustehokkuuden, ketteryyden ja kansalaiskehittämisen [13]. Nämä on kuvattu taulukossa 3. Vastaavia hyötyjä on esitetty muissakin lähdeaineistoissa esimerkiksi Sanchisin et al. tutkimuksessa [40]. Sahinaslan et al. listaa tutkimuksessaan vähäkoodisen ohjelmistokehityksen hyötyjä [2]. Tutkimuksessa todetaan, että jatkuvasti kehittyvässä ja digitalisoituneessa ympäristössä on tärkeää, että yritykset kehittävät uusia ohjelmistoja ja toteuttavat ratkaisuja olemassa oleviin ylläpitoa vaativiin ohjelmistoihin. Aiemmin tunnettujen perinteisten ohjelmistokehitysmenetelmien lisäksi yritysten on tutkittava seuraavan sukupolven vähäkoodisia ohjelmistokehitysmenetelmiä, jotta heidän on mahdollista tuottaa ohjelmistoja nopeasti ja helposti [2]. Vähäkoodinen ohjelmistokehitys tarjoaa oivallisen mahdollisuuden yrityksille, jotka haluavat pysyä kilpailukykyisinä markkinoilla nopeiden ja kustannustehokkaampien ratkaisujen avulla [2].

Taulukko 3: Vähäkoodisen ohjelmistokehityksen hyötyjä.

Kehittämisen nopeus
Kustannustehokkuus
Ketteryys
Kansalaiskehittäjien osallistuminen

Yksi vähäkoodisen ohjelmistokehityksen ohjelmistokehittämisen hyödyistä on kehittämisen nopeus. Forresterin tutkimuksen mukaan vähäkoodiset kehitysalustat nopeuttivat ohjelmistokehitystä 5-10 kertaisesti [40]. Nopeiden kehityssyklien avulla yritys voi nopeasti sopeutua markkinoiden muutoksiin, asiakkaiden muuttuneisiin tarpeisiin tai lainsäädäntöön [13]. Yritys saa julkaistua uusia tai päivitettyjä tuotteita ja palveluita nopeammin eikä kehitykseen mene aikaa kuukausia tai jopa vuosia. Vähäkoodisen ohjelmistokehityksen nopeus perustuu siihen, että suurin osa koodista on valmiiksi kirjoitettua ja koodi on liitetty visuaalisiin komponentteihin [40]. Kehittäjä pystyy hyödyntämään valmiita komponentteja vedä-ja-pudota menetelmällä, jolloin varsinaista koodaamista ei aina tarvita [2].

Vähäkoodisen ohjelmistokehityksen etuna on myös se, että ohjelmistokehitys, automaatio, analytiikka ja botit saadaan käyttöön yhdeltä alustalta [40]. Ohjelmistoja ei rakenneta tyhjästä, sillä ohjelmistokehityksen työkalut ovat osa kehitysalustaa [40]. Näin pystytään keskittymään vain ohjelmistojen kustomointiin ja monimutkaiset taustatyöt voidaan jättää kehitysalustan hoidettavaksi. Vähäkoodiset kehitysalustat saattavat vähentää myös ohjelmistojen ylläpidon tarvetta, kun kehitysalustaan tehty korjaus tai ylläpitotoimi voidaan ottaa kerralla käyttöön kaikkiin sillä toteutettuihin sovelluksiin [22].

Parhaimmillaan vähäkoodinen ohjelmistokehitys voi tuoda helpotusta osaajapulaan, kun liiketoiminnan asiantuntijat ja juniorikehittäjät saavat aikaiseksi valmiita tuotteita [13].

Kansalaiskehittäjien hyödyntäminen onkin vähäkoodisen ohjelmistokehityksen ehdottomia etuja. Kansalaiskehittäjä voi tunnistaa työssään kehittämiskohteita, joita muut eivät huomaa ja joiden automatisointi voi tuottaa isojakin hyötyjä. Kansalaiskehittäjät voivat helpottaa tietohallinnon taakkaa, kun kaikkeen kehittämiseen ei tarvita ohjelmointiosaamista [16, 45]. Vähäkoodiset ohjelmistokehitysalustat ja niiden tarjoamat ominaisuudet ovat kaikkien saatavilla ja täten läpinäkyviä [2]. Lähes kuka tahansa taustastaan ja osaamisestaan riippumatta pystyy käyttämään vähäkoodisia ohjelmistokehitysalustoja ja tästä huolimatta tuotettu koodi on tasalaatuista. Kun kaikilla on mahdollisuudet kehittää ohjelmistoja ilman vahvaa teknistä osaamista, kehitys- ja ylläpitotehtävät voidaan suorittaa sisäisesti tai yhteistyökumppanin toimesta, eikä niitä tarvitse ulkoistaa kolmansille osapuolille [33]. Vähäisen koodin ansiosta vähäkoodiset ohjelmistokehitysalustat mahdollistavat nopean ja selkeän ylläpidon [33]. Kai Lyytinen nostaa kuitenkin Talouselämän blogissaan esiin tärkeän seikan; kansalaiskehittäminen vaatii vahvaa IT:n mukanaoloa [27].

Sanchisin et al. tutkimuksessa viitattiin OutSystemsin vuonna 2019 tekemään 'The State of Application Development' -raporttiin, jossa tutkittiin organisaatioiden perusteita vähäkoodisten kehitysalustojen käyttämiseen [40]. Tutkimukseen osallistuneista 66% piti tärkeimpänä syynä kehittämisen ja reagointikyvyn nopeutta [40]. Seuraavaksi tärkeimmäksi syyksi nousi tarve vähentää riippuvuutta teknisistä osaajista.

3.1.4 Vähäkoodisen ohjelmistokehityksen riskit

Markkinoilla on useita vähäkoodisia kehitysalustoja ja yhtä paljon lupauksia niiden erinomaisuudesta. Tähän uuteen ohjelmistokehitysmenetelmään liittyy myös riskejä, jotka kannattaa huomioida. Tunnistettuja riskejä on koottu taulukkoon 4.

Taulukko 4: Vähäkoodisen ohjelmistokehityksen riskejä.

Sovelluksen päivittämisen haasteet
Virheenkorjaus
Tietoturvariskit
Päätöksenteon perusteen tulkinta ns. black-box

Tisi et al. nostavat esiin kolme vähäkoodisten kehitysalustojen haastetta [43]:

- Skaalautuvuus: vähäkoodiset kehitysalustat on tarkoitettu pääasiassa pienten sovellusten kehittämiseen eikä niiden skaalautuvuutta laajoihin ohjelmistoihin tunneta tarpeeksi
- Vähäkoodisten kehittämismallien pirstaleisuus riippuen kustakin toimittajasta ja sen käyttämistä ohjelmointimalleista
- Kansalaiskehittäjät odottavat, että he osallistuvat kehittämistyöhön ei-teknisellä tasolla ja kehittämisessä käytetään heille tuttuja termejä ja tapoja. Heillä ei ole juurikaan asiantuntemusta ohjelmoinnista.

Vähäkoodisille kehitysalustoille rakennettuja ohjelmistoja voi olla vaikea päivittää, jos niiden kehittämisessä ei ole ollut mukana tietoteknisiä osaajia [49]. Sovelluksen suorituskyky ei välttämättä ole optimaalinen, ja ohjelmistojen integroiminen yrityksen muihin tietojärjestelmiin voi olla vaikeampaa kuin perinteisesti koodattujen järjestelmien. Jotkut online-sovellusgeneraattorit tai mobiilisovellusten luomiseen tarkoitetut vähäkoodisilla alustoilla kehitetyt ratkaisut voivat myös olla alttiita tietoturvariskeille [49].

Ongelmia voi tuottaa myös virheenkorjaus, sillä se saattaa olla vaikeaa kansalaiskehittäjälle, jolla ei ole ohjelmointiosaamista. Myös IT-ammattilaisen voi olla hankalampi tunnistaa virheitä ja etsiä niihin ratkaisuja, kun

kirjoitettua koodia on vähemmän [22]. Vähäkoodiseen ohjelmistokehittämiseen liitetyt ongelmat voivat koskea myös tekoälyn ja ohjelmistojen ymmärrettävyyttä, jos tekoäly tuottaa suurimman osan ohjelmistojen koodista. Tällöin myös ohjelmointia osaavilla ohjelmistokehittäjien voi olla vaikeuksia ymmärtää kyseistä koodia [49]. Koodissa ei välttämättä ole kommentteja tai muuttujien nimet voivat olla epäselviä. Tekoälyn tuottama koodi voi olla vaikeaselkoista.

Lisäksi koodin tuottaneiden algoritmien luontainen black-box -luonne voi synnyttää tilanteita, jossa on usein mahdoton tulkita, miten lopputulokseen on päästy [49]. Black-box -käsitettä käytetään yleisesti erilaisista komponenteista, joista ei saa mitään tietoa niiden sisäisestä suunnittelusta, rakenteesta tai toteutuksesta. Etenkin tekoälystä puhuttaessa black-boxilla tarkoitetaan sitä, ettei käyttäjä saa tietoa sitä, miten järjestelmä on päätynyt tiettyyn lopputulokseen [1].

3.1.5 Vähäkoodisen ohjelmistokehityksen vaiheet

Vähäkoodisia kehitysalustoja käytetään muun muassa liiketoimintasovellusten kehittämiseen. Efimen blogissa Kai Lyytinen listaa asioita, joihin kannattaa kiinnittää huomiota vähäkoodisessa ohjelmistokehityksessä. Ohjelmistojen suunnittelussa tulee huomioida erityisesti alla olevat [28]:

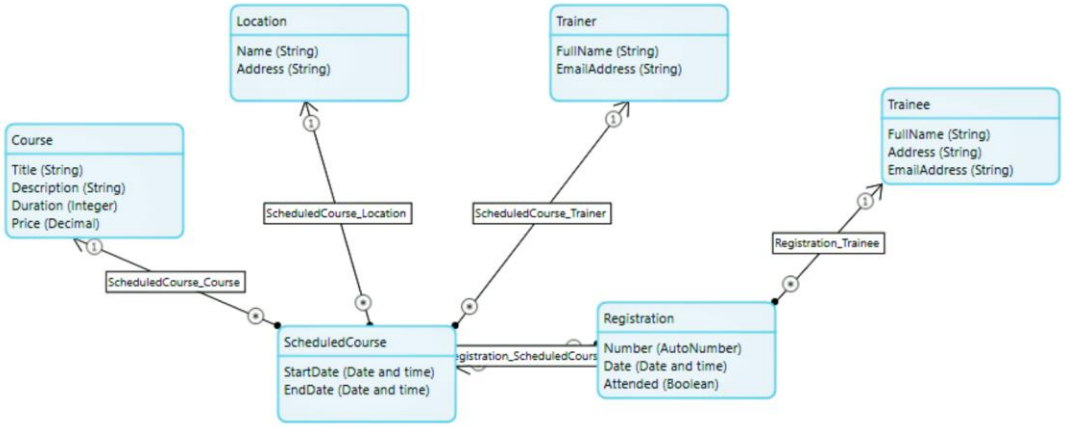
- Ohjelmistossa käytettävien tietojen on oltava kunnossa, samoin kehitettävien sovellusten rooli tiedon käsittelyssä
- Liittymärajapinnat sekä -tekniikat on suunniteltava organisaation sovellusarkkitehtuurin pohjalta
- Käyttöoikeudet ovat vahvassa roolissa; kenellä on oikeus käsitellä, nähdä, luoda ja muokata sovelluksen sisältämää dataa
- Järjestelmän ylläpito tulee huomioida jo kehitysvaiheessa.

Digital Illustrated muistuttaa blogissaan, ettei vähäkoodinen ohjelmistokehitys itsessään poista sovelluskehitysprojektin tarvetta [11]. Tarvetta on edelleen vaatimusten hallinnalle, laadunvarmistukselle ja testaukselle. Sovellusten ylläpidon ja tuen tarve sekä dokumentointi korostuvat ja niihin pitää kiinnittää huomioida jo ohjelmiston kehitysvaiheessa. Myös selkeät roolit ja päätöksentekoprosessit ovat usein välttämättömiä, kun kehittäminen tapahtuu nopeasti [11].

Tyypillisesti sovellusten luonti vähäkoodisten kehitysalustojen avulla voidaan tiivistää kuuteen työvaiheeseen [38]:

1. Tietomallinnus
2. Käyttöliittymä
3. Käyttäjäroolit ja turvamekanismit
4. Työnkulut
5. Integraatiot ulkoisiin tietoihin ja palveluihin
6. Käyttöönotto

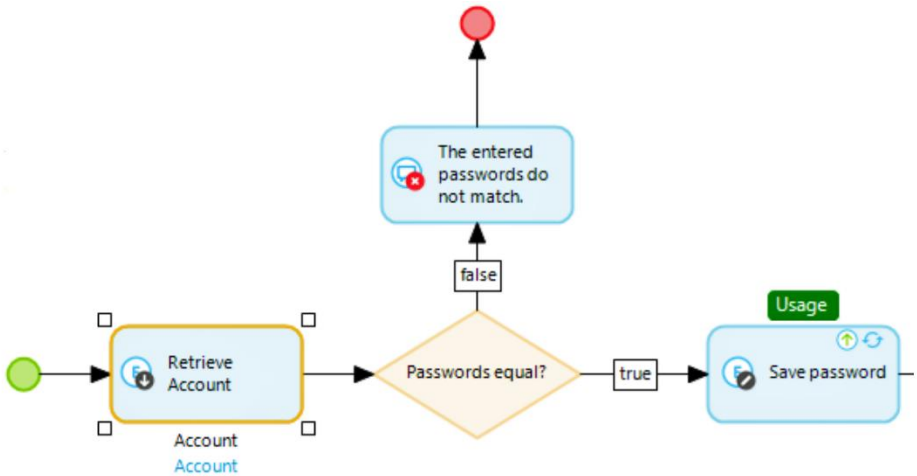
Kun liiketoiminnan tarve ja vaatimukset on tunnistettu ja määritelty, alkaa ohjelmiston kehitystyön ensimmäinen ja kenties vaativin työvaihe eli tietomallinnus [38]. Tietomallinnuksessa kehittäjät määrittelevät tietoskeeman luomalla olioita ja suhteita, sekä määrittelemällä tietojen välisiä riippuvuuksia ja rajoituksia. Tietomallinnus voidaan toteuttaa visuaalisen käyttöliittymän kautta Vedä ja pudota -toimintojen avulla. Tietomallin määrittely on erittäin tärkeä vaihe ja siihen on hyvä käyttää IT-ammattilaisten osaamista. Kuvassa 5 on esimerkki yksinkertaisesta tietomallista, joka on rakennettu Mendixillä. Mendix on yksi niistä vähäkoodisista kehitysalustoista, jota Sahay et al. vertaili tutkimuksessaan [38].



Kuva 5: Mendexillä rakennettu tietomalli [38].

Kun tietomallinnus on valmis, siirrytään ohjelmiston ulkoasun määrittelyyn. Tässä vaiheessa Vedä ja pudota -toiminnot nousevat merkittäväksi tekijäksi toteutuksen nopeuttamista [38]. Käyttöliittymän määrittelyssä käyttäjä luo sovellukselle sivuja ja lomakkeita, joita käytetään myöhemmin näkymien määrittelyyn. Kehitysalustoilla on yleensä erilaisia ulkoasuvaihtoehtoja, joista valitaan sopivin.

Seuraavassa työvaiheessa tehdään käyttäjäroolien määrittely ja hallinnointi sekä turvamekanismien luonti. Seuraava työvaihe on työnkulkujen koordinointi eri lomakkeiden ja sivujen välillä. Myös tähän voidaan käyttää visuaalisia työnkulkuja. Kuvassa 6 on esimerkki työnkulkukaaviosta, joka on kuvattu Mendixissä.



Kuva 6: Mendexillä rakennettu työnkulkukaavio [38].

Mikäli ohjelmisto käyttää ulkoisia tietoja tai palveluita, tulee niiden integrointi tehdä seuraavaksi käyttämällä kolmannen osapuolen sovellusrajapintoja. Vähäkoodisilla kehitysalustoilla toteutetut sovellukset voidaan integroida helposti useisiin sovellusrajapintoihin. Integraatioiden määrittely- ja toteutusvaiheessa on hyvä tehdä yhteistyötä yrityksen sovellusarkkitehtuuria tuntevien osaajien kanssa. Sovellusarkkitehtuurilla tarkoitetaan sovellusten muodostamaa kokonaisuutta ja sovellusten välisiä yhteyksiä [46].

Vähäkoodiset kehitysalustat sisältävät testaustyökaluja, jolla toteutettavaa ohjelmistoa voidaan testata lukuisia kertoja eri työvaiheissa. Kun ohjelmisto on valmis, viimeisenä työvaiheena on ratkaisun siirtäminen tuotantokäyttöön. Useilla vähäkoodisilla kehitysalustoilla tekninen käyttöönotto voidaan tehdä julkaisemalla ohjelmisto muutamalla napsautuksella [38]. Käytännössä käyttöönotto tehdään siten kuin se on yrityksen toimintaohjeissa ja käytänteissä sovittu.

Huomioita tulee kiinnittää myös ohjelmiston elinkaaren hallintaan, jotta ohjelmisto säilyy toimintakykyisenä myös tuotantokäytön aikana. EU muistuttaa tuoreessa asetusehdotuksessaan, että tekoälyä sisältävä ohjelmisto voi sisältää uhkia, joita ei ollut olemassa ohjelmiston kehityksen alussa, vaan niitä on muodostunut kun ohjelmistoa on päivitetty tai kun tekoäly on oppinut koneoppimisen myötä [44].

3.1.6 Vähäkoodisten menetelmien käyttökohteet

Tässä kappaleessa pohditaan sitä, millaiseen kehittämiseen vähäkoodista ohjelmistokehitystä kannattaa käyttää. Tutkimusnäyttöä parhaista käyttökohteista ei ole, mutta internetistä löytyy asiaa käsitteleviä sivustoja. Tero Tapanainen pohtii eCraftin blogissa, että vähäkoodinen ohjelmistokehitys sopii parhaiten sellaiseen ohjelmistokehitykseen, jossa ohjelmiston ominaisuuksien määrä on suhteellisen rajattu [13]. Blogin mukaan vähäkoodisesta ohjelmistokehityksestä hyötyy eniten, kun ohjelmiston monimutkaisuusaste ei ole suuri, mutta tekemisen volyymiä on paljon.

Solitan blogin mukaan vähäkoodinen ohjelmistokehitys soveltuu muun muassa liiketoimintaprosessien digitalisointiin, sähköisten palveluiden ja lomakkeiden kehitykseen sekä nopeisiin kokeiluihin [42]. Sahay et al. toteaa tutkimuksessaan, että vähäkoodiset kehitysalustat sopivat organisaatioille, joilla on rajatut IT-resurssit ja budjetti [38]. Tämä siksi, että vähäkoodisilla kehitysalustoilla pystytään kehittämään toimivia ohjelmistoja nopealla aikataululla. Esimerkkinä tutkimus mainitsee CRM-sovellukset.

Vähäkoodinen ohjelmistokehitys lupaa paljon, mutta se ei ole patenttiratkaisu kaikkiin liiketoiminnan tarpeisiin ja ongelmiin. Tehokkaimmat ratkaisut saavutetaan yleensä silloin, kun hyödynnetään sekä vähäkoodista ohjelmistokehitystä että “pro-codea” [11]. Näin varsinkin silloin, kun ongelma on monimutkainen. Pro-codella tarkoitetaan koodipohjaista ohjelmointia [11].

3.2 Tekoälyn ymmärrettävyys ja selitettävyys

3.2.1 Mitä on tekoälyn ymmärrettävyys ja selitettävyys

Toinen tutkimuskysymys on, miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta. Tässä luvussa pyrin selvittämään, mitä selitettävyys ja ymmärrettävyys tekoälyjärjestelmissä tarkoittavat ja mitä käsitteet merkitsevät tilaajan näkökulmasta. Tekstissä esiintyvät termit ihminen ja käyttäjä, ja niillä tarkoitetaan tilaajaa.

Tekoäly

Tekoälyn määritelmä on hyvin laaja eikä sille ei ole olemassa yhtä yleisesti hyväksyttyä määritelmää. Euroopan komissio on kuitenkin tehnyt aloitteen tekoälyn käsitteen muotoilemiseksi ja sääntelemiseksi. Heidän tekoälyä käsittelevä korkean tason asiantuntijaryhmänsä määrittelee tekoälyn

seuraavasti: “Artificial intelligence (AI) systems are software (and possibly also hardware) systems designed by humans that, given a complex goal, act in the physical or digital dimension by perceiving their environment through data acquisition, interpreting the collected structured or unstructured data, reasoning on the knowledge, or processing the information, derived from this data and deciding the best action(s) to take to achieve the given goal. AI systems can either use symbolic rules or learn a numeric model, and they can also adapt their behavior by analyzing how the environment is affected by their previous actions.

As a scientific discipline, AI includes several approaches and techniques, such as machine learning (of which deep learning and reinforcement learning are specific examples), machine reasoning (which includes planning, scheduling, knowledge representation and reasoning, search, and optimization), and robotics (which includes control, perception, sensors and actuators, as well as the integration of all other techniques into cyber-physical systems).” [36]

Tekoälyllä tarkoitetaan siis laitteiden, yleensä tietokoneiden, kykyä jäljitellä ihmisen älykkyyttä [29]. Tällä älykkyydellä tarkoitetaan yleisesti koneen kykyä kommunikoida, järkeillä ja toimia itsenäisesti niin tutuissa kuin uusissa tilanteissa [12]. Tekoälyn perimmäisenä tavoitteena on luoda tietokoneohjelmia, jotka pystyisivät ratkaisemaan ongelmia ja saavuttamaan niille asetettuja tavoitteita yhtä hyvin kuin ihmiset [29]. Niin sanottua täydellistä älykkyyttä ei kuitenkaan ole vielä pystytty saavuttamaan [12] vaan nykyisesti tunnetut parhaatkin tekoälyjärjestelmät luokitellaan olevan vain "jokseenkin älykkäitä" [29]. Älykkyyteen liittyy siis monia mekanismeja, joita monia ei ole vielä pystytty jäljittelemään ja toteuttamaan [29].

Tekoälylle on asetettu kunnianhimoiset tavoitteet, tosin monien tutkijoiden mielestä ihmisen tasoisen älykkyyden tavoitteleminen ei ole edes mahdollista [29]. Kun puhutaan tekoälystä, sitä käytetäänkin usein koneoppimisen synonyyminä. Koneoppimisella tarkoitetaan algoritmeja ja tilastollisia malleja, jotka pystyvät tunnistamaan ja päättelemään uusia malleja olemassa olevasta datasta. Koneoppiminen on nykyisin yleisin tekoälyn muoto, joka on ohjelmoitu oppimaan joko ihmisten toimesta tai itsenäisesti harjoitusdatan avulla [12]. Koneoppimismalli käyttää erilaisia menetelmiä oppiakseen ja analysoidakseen annettua tietoaineistoa. Koneoppimismalleja voidaan käyttää tulevaisuuden ennustamiseen tai menneisyyden ymmärtämiseen [4]. Vaikka koneoppiminen on yksi tekoälyn muodoista, tekoälyjärjestelmä voidaan kuitenkin luoda ilman koneoppimisalgoritmeja, joten nämä kaksi termiä eivät ole synonyymejä.

Selitettävyys

Selitettävyydellä tarkoitetaan kykyä selittää ja oikeuttaa päätökset järjestelmän käyttäjälle. Selitettävyys on käsite, joka on monien aktiivisen tekoälytutkimuksen alojen risteyskohdassa, ja siinä korostuvat avoimuus, kausaalisuus, puolueellisuus, oikeudenmukaisuus ja turvallisuus [21]. Tekoälymallien odotetaan olevan selitettäviä, eli sen tekemät päätökset on mahdollista selittää ymmärrettävällä tavalla ihmiselle [23].

Läpinäkyvyys

Läpinäkyvyydellä tarkoitetaan sitä, että meillä on oikeus saada tietoa meitä koskevista päätöksistä käsittämillämme termeillä, muodoilla ja kielillä [21].

Ymmärrettävyys

Ymmärrettävyydellä tarkoitetaan kykyä ymmärtää tekoälyjärjestelmän toimintaa [15]. Ymmärrettävyys vahvistaa myös tekoälyn oikeudenmukaisuutta ja vastaa kysymyksiin ”miten tekoäly toimii” ja ”kuka on sen toiminnasta vastuussa” [15]. Tekoälyjärjestelmien osalta tämä on ensisijaisen tärkeää, kun tiedon käsittelijänä on ihmisen sijaan kone.

3.2.2 EU:n uudet säädökset luotettavalle tekoälylle

Tekoälyn luotettavuutta pyritään parantamaan myös lainsäädännöllä. EU:n komissio antoi huhtikuussa 2021 ehdotuksen Euroopan parlamentin ja neuvoston asetukseksi tekoälyn eurooppalaisesta lähestymistavasta (COM/2021/206 final). Samassa yhteydessä komissio antoi tekoälyn tiedonannon (COM/2021/205 final) sekä sen liitteenä päivitetyn tekoälyn koordinoidun toimintasuunnitelman. Asetusehdotus pohjautuu komission 19.2.2020 antamaan tiedonantoon Euroopan digitaalisesta tulevaisuudesta sekä tekoälyä koskevaan valkoiseen kirjaan (E24/2020vp).

EU korostaa luottamuksen ekosysteemin tärkeyttä eli sitä, että tekoälyn tulee pohjautua EU:n säädöksiin, arvoihin ja perusoikeuksiin. Lähtökohta on, että tietojärjestelmän käyttäjän tulee ymmärtää, milloin hän on vuorovaikutuksessa tekoälyjärjestelmän kanssa. Lisäksi asetusehdotuksessa huomioidaan tietyt käyttötavat, kuten sisältöjen manipulointi, joista tulee ilmoittaa käyttäjälle. Asetusehdotuksessa tekoälyjärjestelmät luokitellaan neljään riskiryhmään käyttötarkoituksen pohjalta. Tekoälyjärjestelmien sääntely määräytyisi sen perusteella, missä määrin tekoälyjärjestelmän käyttö voisi aiheuttaa merkittävää haittaa ihmisten terveyteen, turvallisuuteen tai perusoikeuksien

toteutumiseen. Asetusehdotus jakaa tekoälyjärjestelmät neljään riskiryhmään:

1. Riskit, jotka eivät ole hyväksyttävissä
2. Korkeat riskit
3. Vähäiset riskit
4. Minimaaliset riskit

Ensimmäisen riskiryhmän riskit olisivat kokonaan kiellettyjä tiettyjä poikkeuksia lukuun ottamatta. Tällaisiksi tekoälyjärjestelmiksi katsotaan sellaiset ratkaisut, joita pidetään selvänä uhkana ihmisten turvallisuudelle, toimeentulolle tai oikeuksille. Kiellettyjä tekoälyjärjestelmiä olisivat esimerkiksi sellaiset ratkaisut, joilla valtiot pisteyttävät kansalaisiaan sosiaalisten tekijöiden perusteella [31].

Korkeariskiset tekoälyjärjestelmät sallittaisiin ainoastaan tiukkojen vaatimusten täyttyessä ja ehdotuksessa niiden tarjoajille on määrätty velvollisuuksia. Korkeariskisten tekoälyjärjestelmien on muun muassa oltava erittäin tarkkoja, vakaita ja tietoturvallisia, järjestelmissä käytettävän datan on oltava laadukasta riskien ja syrjivien tulosten minimoimiseksi, järjestelmät on dokumentoitava tarkasti sekä käyttäjille on annettava selkeät ja riittävät tiedot järjestelmän toiminnasta ja käyttötarkoituksesta. Lisäksi EU vaatii, että ihmisen on kyettävä valvomaan korkeariskisen tekoälyjärjestelmän toimintaa riittävästi riskien minimoimiseksi. Ihmisen tulee pystyä keskeyttämään välittömästi korkeariskisen tekoälyjärjestelmän käyttö ja ihmisellä tulee olla mahdollisuus jättää huomiotta, korjata, ohittaa tai peruuttaa korkeariskisen tekoälyjärjestelmän tuotos. Asetusehdotuksessa huomioidaan myös se, että korkean riskin tekoälyjärjestelmän on toimittava kestävyyden, tarkkuuden ja tietoturvallisuuden suhteen yhdenmukaisesti koko elinkaarensa ajan. Korkean riskin tekoälyjärjestelmiä olisivat esimerkiksi järjestelmät, joita käytetään rekrytoinnissa työhakemusten karsimiseen tai lainansaannissa luottokelpoisuuden pisteytykseen tai järjestelmät, jotka voivat vaarantaa ihmisten hengen ja terveyden liikenteenohjauksessa [31].

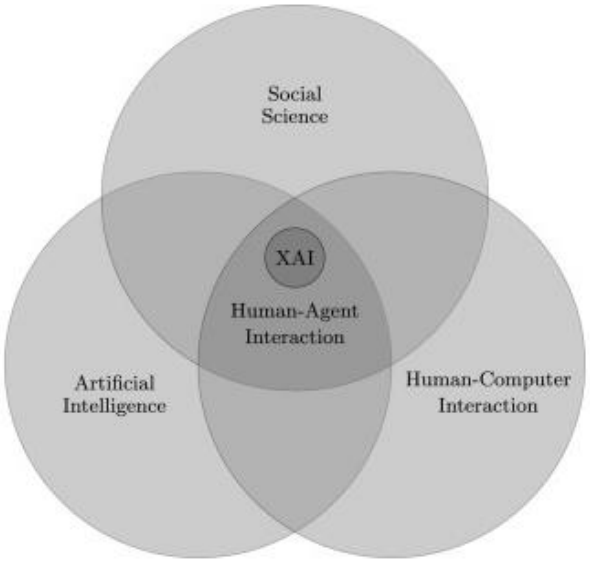
Vähäisten riskien järjestelmien kohdalla sen sijaan riittäisi, että käyttäjä tietää tekoälyn olemassaolosta. Vähäisen riskin tekoälyjärjestelmiä ovat muun muassa asiakaspalvelun chat-robotit. Näissä tavoite olisi, että käyttäjä ymmärtäisi milloin ihminen on tekemisissä koneen kanssa. Riskejä, jotka aiheuttavat minimaalisen riskin, asetusehdotus ei juurikaan säätele. Näihin kuuluisivat esimerkiksi roskapostifiltterit [31].

Ennen asetusehdotuksen voimaantuloa, EU-parlamentin ja jäsenvaltioiden pitää hyväksyä se tavanomaisessa lainsäätämisjärjestyksessä. Kun asetus on hyväksytty, sitä sovelletaan kaikkialla EU:ssa. Tekoälyyn liittyvien asetusten olisi arvioitu tulevan voimaan vuonna 2022 ja niitä alettaisiin soveltaa siirtymäkauden jälkeen aikaisintaan vuonna 2024.

EU pyrkii ohjeillaan edistämään luotettavaa tekoälyä [14]. Selitettävyys on keskeinen asia käyttäjien tekoälyjärjestelmiä kohtaan tunteman luottamuksen luomiseksi ja ylläpitämiseksi. Prosessien tulee olla avoimia, tekoälyjärjestelmien tarkoitus on ilmoitettava avoimesti ja päätökset on pystyttävä selittämään niille, joihin ne suoraan tai välillisesti vaikuttavat [14]. Käyttäjän tulee voida ymmärtää, milloin hän on vuorovaikutuksessa tekoälyn kanssa. Lisäksi asetusehdotuksessa huomioidaan tietyt käyttötavat, kuten sisältöjen manipulointi, joista tulee ilmoittaa käyttäjälle [31]. Käyttäjä voi tuntea olonsa huijatuksi, jos hän perustaa päätöksensä virheelliseen käsitykseen toimijan identiteetistä. Käyttäjälle tulee ilmoittaa, jos esimerkiksi henkilö, esine tai paikka voi vaikuttaa aidolta ja käyttäjä voi erehtyä luulemaan sitä todelliseksi.

3.2.3 Selittävä tekoäly

Tässä kappaleessa pohditaan, mitä tarkoitetaan selittävällä tekoälyllä. Käsitteenä selittävällä tekoälyllä (XAI) tarkoitetaan tutkimusalaa, jonka tavoitteena on tehdä tekoälyn tekemistä päätöksistä käyttäjälle ymmärrettävämpiä [1]. Adadin ja Berradan mukaan XAI:n tavoitteiden saavuttaminen on mahdollista vain eri tieteenalojen yhteistyöllä [1]. Selittävän tekoälyn osa-alueella on käynnissä suuri määrä tutkimuksia eri tieteiden aloilla. Miller on tutkinut XAI:ta yhteiskuntatieteiden näkökulmasta. Hän määrittelee XAI:n tekoälyn, yhteiskuntatieteen sekä ihmisen ja tietokoneen välisen vuorovaikutuksen leikkauspisteeksi [30]. Tämä on esitetty kuvassa 7. Selittävään tekoälyyn liittyvän tutkimuksen ongelmana on, ettei terminologiaa ole vahvistettu ja eri tieteenalat ja tutkimusryhmät käyttävät samankaltaisista käsitteistä eri nimityksiä.

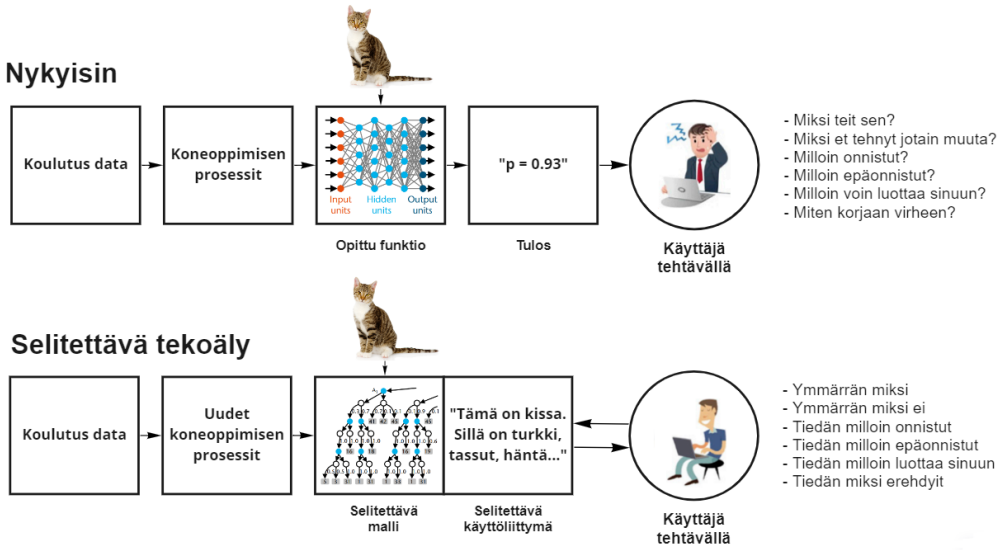


Kuva 7: Selittävä tekoäly Millerin määritelmänä [30].

Selitettävyys on osa tavoitetta, jossa tekoälyä hyödyntäviä järjestelmiä kehitetään vastuullisella tavalla [14]. Vastuullisuudella tarkoitetaan sitä, että kyseisten järjestelmien mahdollisesti aiheuttamat riskit otetaan etukäteen huomioon ja niitä pyritään aktiivisesti välttämään. Järjestelmien ja niiden sisäisen toiminnan mallit edesauttavat riskien huomioimisen ja aktiivisen välttämisen. Valitettavasti tekoälyalgoritmien ongelmana on, että niiden sisäisestä toiminnasta on vaikea saada tietoa [1] ja monimutkaisten tekniikoiden, kuten syväoppimisen, käyttö tekee tekoälyn päätöksistä ihmisille vaikeasti ymmärrettäviä [41]. Tekoälyn päätöksentekoprosessi jää usein selittämättömäksi, jonka takia käyttäjien on vaikea tietää, miten tekoäly on päätynyt tiettyihin päätöksiin, suosituksiin, ennusteisiin tai toimenpiteisiin. Tämän selitettävyysongelman ratkaisemiseksi on luotu malleja ja menetelmiä, jonka tavoitteena on pyrkiä ratkaisemaan läpinäkyvyyteen liittyviä ongelmia ja tehdä tekoälyjärjestelmien tuloksista ymmärrettävämpiä [1].

Selitettävän tekoälyn luominen ei ole yksinkertaista myöskään sen takia, että se on käsitteenä epäselvä. Ei ole olemassa standardia tai yleisesti hyväksyttyä kuvausta sille, mitä tarkoitetaan selittävällä tekoälyllä [1]. Defense Advanced Research Projects Agencyn (DARPA) mukaan selitettävä tekoäly pyrkii tuottamaan selitettäviä malleja, joiden avulla käyttäjä kykenee ymmärtämään, luottamaan ja hallitsemaan tekoälymalleja [20]. Nykytiedon mukaan ihmiset luottavat ja hyväksyvät algoritmien tekemät päätökset, mikäli järjestelmät ovat selitettävissä ja ymmärrettävissä [47]. Selittävän tekoälyn mallissa koneet kykenevät hahmottamaan missä ympäristössä ja kontekstissa ne toimivat [20].

Kuvassa 8 on Gunningin XAI-ohjelman keskeiset tavoitteet. Näitä ovat, että käyttäjä ymmärtäisi miten tekoälysovellukset toimivat, mihin päätökset perustuvat ja missä mahdolliset virheet syntyvät [20]. Nykytilassa tekoälysovellukset näyttäytyvät käyttäjille mustina laatikkoina.



Kuva 8: XAI-ohjelman keskeiset tavoitteet liittyvät läpinäkyvyyden lisäämiseen [20].

Tekoälyn selitettävyys ei kuitenkaan ole tärkeää pelkästään päätösten perustelemiseksi, vaan sen avulla pystytään myös ehkäisemään, ettei järjestelmä toimi väärin [39]. Järjestelmän ymmärtäminen mahdollistaa näkyvyyden tuntemattomiin haavoittuvuuksiin sekä virheisiin, joiden avulla virheitä pystytään korjaamaan ajoissa [1]. Virheiden löytämisen lisäksi selitettävyys helpottaa myös järjestelmän kehitystyötä, sillä selitettävää ja ymmärrettävää tekoälyä on helpompi ylläpitää ja kehittää, kun käsiteltävästä ongelmasta on mahdollista saada uutta tietoa [39]. Paremmat selitykset voivat myös parantaa olemassa olevia malleja ja avata uusia mahdollisuuksia [41]. Selitettävyyden perusteella kehittäjien on mahdollista ymmärtää, miten päätöksentekomallia on muutettava, jotta tulevaisuudessa saadaan haluttu lopputulos [47].

Vaikka selitettävyyden takaamiseksi on tarjottava personoituja selityksiä [41], on selitettävyyden tuominen tekoälyjärjestelmiin kuitenkin suuri teknninen haaste [1]. Tekoälyjärjestelmät ja niiden hyödyntävät ML-algoritmit ovat monimutkaisia, jonka takia niiden tuottamat tulokset ovat useasti hyvin vaikeita ymmärtää. Monimutkaisuutta on kuitenkin lähes mahdoton poistaa, sillä juuri se antaa ML-algoritmeille poikkeuksellisen toimintakyvyn [1]. Jotta selitettävyyteen pystytään, tarvitaan kehittynyttä tiedon louhintaa ja preferenssien esiin nostamista, jotta raakadatasta saadaan haluttua selitettävää tietoa [41].

Selitettävä tekoäly mahdollistaa sen, että tekoälyn tuottama lopputulos tai sen tekemä päätös pystytään perustelemaan. Tämä on erityisen tärkeää tilanteissa, joissa tekoälyn tekemät päätökset ovat odottamattomia [1], tällöin on hyvin tärkeää pystyä informoimaan kyseisen päätöksen perusteluista ja selittää päätöksen syyt [47]. Perustelut lisäävät päätösten oikeudenmukaisuutta ja eettisyyttä, sekä todentavat käyttäjälle, että ohjelmisto toimii lainsäädännön mukaisesti [1].

Kaikki tekoälyjärjestelmät eivät kuitenkaan ole vaikeaselkoisia tai mahdottomia ymmärtää. Riippuen tekoälyn tarkkuudesta, toisia algoritmeja on helpompi tulkita kuin toisia. Kun tekoälyjärjestelmiä rakennetaan, tulee usein tilanne, jossa joudutaan tekemään kompromisseja tekoälyn tarkkuuden ja tulkittavuuden välillä [1]. On myös tilanteita, joissa selitettävyyttä ei edes odoteta. Tällaisia tilanteita on järjestelmät, joilla ei ole merkittäviä seurauksia väärästä päätöksestä tai tilanteet, jossa käyttäjät luottavat järjestelmään, vaikka sen tiedetään olevan epätäydellinen [23].

Yhteenvetona voidaan todeta, että tekoälyjärjestelmän luotettavuuden lisäämiseksi on pystyttävä ymmärtämään, miksi tekoälyjärjestelmä käyttäytyi tietyllä tavalla ja miksi se päätyi tiettyyn tulokseen. Selittävän tekoälyn tavoitteena on tarjota selitys tekoälyn tekemille päätöksille, selittää järjestelmän käyttäytymistä ja antaa ohjeita järjestelmän tekemien virheiden ehkäisemiseksi [1]. XAI-tutkimusala tutkii näitä asioita, jotta voidaan sekä selittää tekoälyjärjestelmän käyttäytymistä että ottaa käyttöön luotettavaa teknologiaa.

3.2.4 Tekoälyn ja ihmisen yhteistyö

Päätöksenteko vaatii usein sekä tietojen keruuta ja käsittelyä että intuitiivista ajattelua. Tässä kappaleessa avataan tekoälyn ja ihmisen päätöksenteon eroja sekä pohditaan, miten näiden lähestymistapojen yhteistyötä voisi hyödyntää. Lisäksi pohditaan, miksi ymmärrettävyys ja selitettävyys päätöksenteossa ovat niin tärkeitä.

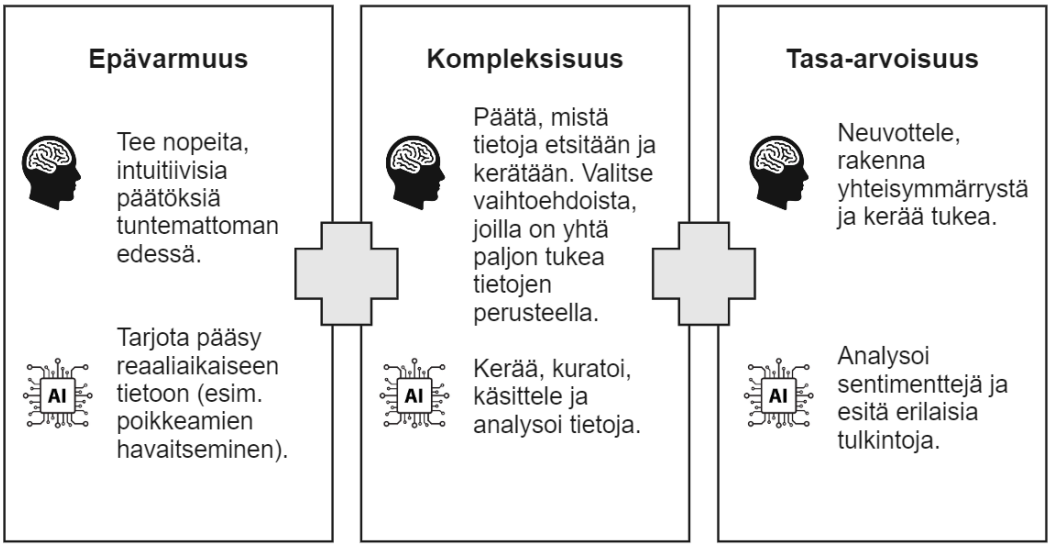
Tekoäly kykenee analysoimaan nopeasti monimutkaisia ja runsaasti muuttujia sisältäviä tietomassoja. Tämän mahdollistaa tekoälyn ylivoimainen kvantitatiivinen, laskennallinen ja analyyttinen tehokkuus [26]. Tekoäly voi auttaa ihmisiä päätöksenteossa luomalla uusia ideoita todennäköisyys- ja datalähtöisten laskelmien avulla sekä tunnistamalla eri tekijöiden välisiä suhteita [26]. Adadin ja Berradan mukaan pelkkä tekoälymallin selittäminen ei riitä, vaan ihmisen on ymmärrettävä sen toiminta [1]. Selitettävyys voi tapahtua vain vuorovaikutuksessa ihmisen ja koneen välillä.

Ashoorin ja Weiszin tutkimuksessa tutkittiin osallistujien luottamusta tekoälyn tekemiin päätöksiin [6]. Tutkijoiden mukaan luottamus koostuu yleisestä luottamuksesta prosessiin, tulosten luotettavuudesta, tekoälyn teknisestä pätevyydestä, tekoälyn ymmärrettävyydestä ja osallistujien henkilökohtaisesta sitoutumisesta prosessiin. Tutkimuksessa osallistujat ilmaisivat ristiriitaisia huolenaiheita empatian puutteesta tekoälyn tekemässä päätöksenteossa [6]. Joidenkin osallistujien mielestä oli hyvä, että tekoälyltä puuttuvat inhimilliset tunteet, sillä se lisää luottamusta prosessiin. Nämä osallistujat suhtautuivat positiivisesti siihen, että tekoäly tekee päätökset tunteettomasti perustuen tosiasioihin. Toiset osallistujat suhtautuivat varauksellisesti tekoälyn empatian puutteeseen ja olivat sitä mieltä, ettei tekoälyä pitäisi käyttää sellaiseen päätöksentekoon, jossa tarvitaan tulkintaa. Vaikka empatian puutteesta ollaan ristiriitaisia, tutkimuksen perusteella voidaan todeta, että tulkittavien mallien käyttö ja ymmärrys siitä, miten tekoälymalli on koulutettu ja testattu, parantavat merkittävästi ihmisten luottamusta tekoälyn tekemiin päätöksiin [6].

Jarrahi on tutkinut tekoälyn ja ihmisen välistä yhteistyötä päätöksenteossa, ja näkee siinä paljon mahdollisuuksia [26]. Yhteistyö tuottaa tulosta, kun tekoälyn nopeus tiedon keräämisessä ja analysoinnissa yhdistetään ihmisen intuitiiviseen harkintakykyyn ja oivallukseen. Jarrahi nostaa esiin esimerkin, jossa tutkittiin imusolmukesyövän tunnistamista kuvista. Tekoälyn virheprosentti oli 7,5 % ja patologien 3,5 %. Kun sekä tekoälyn että patologien tekemät havainnot yhdistettiin, yhteistyö johti 85 prosentin virhevähennykseen.

Ihmisen tekemät päätökset eivät perustu ainoastaan tietojen keruuseen ja käsittelyyn, vaan päätökset syntyvät alitajunnassa intuition piirissä [10]. Esimerkiksi organisaatioiden johtajien päätöksenteko vaatii tiedon käsittelyn lisäksi emotionaalista ja sosiaalista älykkyyttä, jotta työntekijät ja ulkoiset sidosryhmät saadaan vakuutettua päätösten tarpeellisuudesta [26]. Tämä on suurimpia eroja tekoälyn ja ihmisen päätöksenteossa, sillä tekoälyllä ei ole käytössä tätä "tunneälyä", vaan tekoälyn päätökset perustuvat vain loogiseen päättelyyn [26]. Tekoälyn objektiivinen ja persoonaton lähestymistapa voi olla jopa ristiriidassa monien organisaatioissa tehtyjen intuitiivisten päätösten [26].

Ihmisen ja tekoälyn päätöksenteon yhteistyö voi toteutua kahdella tavalla [26]. Ihminen ja tekoäly voivat kumpikin keskittyä siihen lähestymistapaan, jossa ne ovat parhaimmillaan. Tekoäly käyttää analyyttistä lähestymistapaa ja ihminen intuitiivistä. Toinen vaihtoehto, erityisesti monimutkaisessa päätöksenteossa, on että päätökset tehdään yhdessä. Ihmisen ja tekoälyn päätöksenteon yhteistyötä on esitetty kuvassa 9.



Kuva 9: Ihmisten ja tekoälyn yhteistyö epävarmoissa, monimutkaisissa ja epäselvissä päätöksentekotilanteissa [26].

Myös EU korostaa asetusehdotuksessaan läpinäkyvyyden tärkeyttä tekoälyjärjestelmien suunnittelussa ja kehittämisessä [44]. Jotta käyttäjät ymmärtäisivät tekoälyjärjestelmien toiminnan ja pystyisivät arvioimaan niiden tekemiä lopputuloksia, on tärkeää tavoitella mahdollisimman ymmärrettävää tekoälyä. Ymmärrettävyys lisää tekoälyjärjestelmien läpinäkyvyyttä [44]. Jos tekoälyjärjestelmä vaikuttaa merkittävästi ihmisten elämään, pitää sen päätöksentekoprosessista antaa asianmukainen selitys [14]. Tällainen selitys tulee antaa siten, että päätökseen johtavat syyt ovat ymmärrettävissä kyseisen sidosryhmän (esimerkiksi maallikon tai viranomaisen) asiantuntemuksella.

3.2.5 Tekoälyn päätöksenteon puutteet

Tekoälyn tekemässä päätöksenteossa nähdään monia epäkohtia. Tekoälyn analyyttinen lähestymistapa päätöksenteossa perustuu sille opetettuun tietoon [26]. Jos jotain tietoa ei ole tekoälylle opetettu, se ei osaa käsitellä sitä. Tekoäly ei myöskään kykene ottamaan huomioon erityisiä olosuhteita, joihin sitä ei ole koulutettu [6].

Tekoälyn päätöksenteko voi perustua historialliseen dataan ja tällöin on vaarana, että historia toistaa tekemiään virheitä [6]. Ashoorin ja Weiszin tutkimuksessa osallistujat ilmaisivat huolensa vääristyneistä ja puolueellista tiedoista, erityisesti silloin, kun on kyse päätöksistä, joilla on suuri merkitys ihmisen elämään [6]. Toisaalta jotkut osallistuvat pitivät tekoälyä puolueettomana ja oikeudenmukaisena päätöksentekijänä ihmiseen verrattuna.

Ihmisiin verrattuna tekoäly ei pysty käsittelemään ongelmaa kokonaisvaltaisesti tai maalaisjärjellä. Samoin voi käydä, jos ympäristö on kovin epävarma tai monimutkainen [26]. Tämä ongelma nousee esiin etenkin määritellyn tietämysalueen ulkopuolella [8].

Tekoälyllä on lähes mahdotonta simuloida epätavanomaisia ja luovia päätöksentekotilanteita, sillä "koneet eivät pysty vangitsemaan ihmisen intuition sisäistä logiikkaa ja alitajuisia malleja" [26]. Sama koskee tilanteita, joista ei ole olemassa ennakkotapauksia [26]. Tekoälyn päätöksenteossa yksi suurimmista ongelmista on epävarmuus [26]. "Tekoäly ja muut älykkäät teknologiat voivat tuottaa uusia ideoita todennäköisyys- ja tietopohjaisten tilastollisten päättelymenetelmien avulla, ja tekoälyn ainutlaatuiset mahdollisuudet tunnistaa monien tekijöiden välisiä suhteita voivat auttaa inhimillisiä päätöksentekijöitä keräämään ja käyttämään tehokkaammin uutta tietoa" [26].

Tekoälyllä saadaan koneet suoriutumaan tehtävistä, jotka vaativat suurta laskentatehoa [21]. Käyttäjille on tärkeää ymmärtää, miksi tekoäly päätyi tiettyyn päätökseen. Käyttäjät luottavat tekoälyn tekemiin päätöksiin silloin, kun päätös perustuu ymmärrettävissä oleviin päätöksentekomalleihin [6]. Luottamus tekoälyn käytöstä vähenee huomattavasti, kun tekoälyä ei pysty tulkitsemaan, vaan se käyttää ns. black-box malleja [6]. Black-box mallit tuottavat lopputuloksia syötteiden perusteella ilman että ne pystyvät selittämään, miksi järjestelmä päätyi tiettyyn lopputulokseen [30]. Ne esittävät ainoastaan lopputuloksen, jonka käyttäjä voi joko hyväksyä tai hylätä [19]. Ashoorin ja Weiszin tutkimuksessa huomattiin, että osallistujat eivät luottaneet mustan laatikon mallien käyttöön yhtä paljon kuin malleihin, joiden logiikan he ymmärsivät [6]. Pieters toteaa tutkimuksessaan, ettei tekoälyjärjestelmien black-box malli ole aina ongelmallinen [30]. Jos käyttäjät luottavat järjestelmän tekemiin päätöksiin, he eivät välttämättä ole kiinnostuneita sen toiminnasta. Selitystä vaaditaan silloin, jos käyttäjä epäilee tekoälyn päätöstä [30].

Yhteenvetona voidaan todeta, että tekoälyjärjestelmien tehokkuutta rajoittaa koneen kyvyttömyys selittää päätöksiään ja toimiaan järjestelmän käyttäjille [26]. Selitettävyys on olennaisen tärkeää, jotta käyttäjät voivat ymmärtää, luottaa ja hallita tekoälyjärjestelmiä [20].

3.2.6 Miten parantaa tekoälyn ymmärrettävyyttä ja selitettävyyttä

Tässä kappaleessa etsitään keinoja siihen, miten tekoälyn ymmärrettävyyttä ja selitettävyyttä voitaisiin parantaa. Jos keinoja löytyy, näihin kannattaa panostaa järjestelmän koko elinkaaren ajan.

Luottamus tekoälyyn riippuu siitä, kuinka hyvin käyttäjät ymmärtävät tekoälyn päätöksentekomallia [6]. Ashoorin ja Weiszin tutkimuksessa todettiin, että osallistujat luottivat tekoälyn tekemiin päätöksiin selvästi enemmän, kun he ymmärsivät miten päätöksentekomalli luotiin ja koulutettiin [6]. Ihmisen osallistumista päätöksentekoon pidettiin oleellisena luottamuksen kannalta. Ihmisen tulee olla mukana joko päätöksenteossa tai vähintään tekoälyn kouluttajana. Mallia, jossa tekoäly kouluttaa itsensä, pidettiin epäluotettavana. Tutkimuksen lopputuloksena todetaan, että luotettavan tekoälyn kehittäminen edellyttää suurta panostusta sekä tekoälyn kehittäjiltä, sitä käyttäviltä ihmisiltä ja niiltä ihmisiltä, joihin tekoälyn tekemät päätökset vaikuttavat [6]. Myös Pieters tarkastelee tekoälyjärjestelmien vastuita [30]. Jos käyttäjä saa riittävän selityksen tekoälyjärjestelmän tekemästä päätöksestä ja voi luottaa siihen, käyttäjä voi jakaa vastuun järjestelmän käytön seurauksista. Pieters toteaa, että tekoälyjärjestelmien suunnittelijat ovat vastuussa järjestelmien suunnittelusta siten, että käyttäjien pitää pystyä luottamaan niihin.

Myös EU kiinnittää huomiota tekoälyn kouluttamiseen. Se esittää asetusehdotuksessaan, että korkean riskin tekoälyjärjestelmiin, jotka sisältävät tietomallien kouluttamista, on sovellettava asianmukaisia tiedonhallintakäytäntöjä [44]. Käytännöillä tarkoitetaan muun muassa asianmukaista tiedonkeruuta ja tietojen valmistelua kuten tietojen merkintää, puhdistusta, rikastamista ja yhdistämistä. Lisäksi koulutus-, validointi- ja testaustietoaineistojen on oltava asianmukaisia, edustavia ja virheettömiä [44]. EU:n mukaan tekoälyjärjestelmän päätöksen tuottavat tiedot ja algoritmit tulee dokumentoida mahdollisimman hyvin läpinäkyvyyden lisäämiseksi. Tämä auttaa tunnistamaan päätöksentekoon vaikuttavat syyt, erityisesti silloin kun tekoälyjärjestelmä teki virheellisen päätöksen.

Luottamus tekoälyyn kasvaa, jos tekoälyjärjestelmä antaa käyttäjälle selityksen, joka perustelee sen tekemän suosituksen, päätöksen tai toimenpiteen. Erityisen tärkeänä luottamus nähdään silloin, jos ympäristö on epävarma tai monimutkainen [26]. Tärkeitä seikkoja luottamuksen lisäämiseksi ovat

- yksittäisten päätösten ymmärtäminen
- kokonaisratkaisun ymmärtäminen
- mahdollisuus ymmärtää ja arvioida vahvuuksia ja heikkouksia
- virheiden tunnistaminen
- vaihtoehdot virheiden korjaamiseen [20].

Joidenkin tekoälyalgoritmien käyttökelpoisuutta voidaan parantaa, jos päätöksentekoprosessissa on ihminen mukana [26]. Tekoäly tuskin pystyy oppimaan tai jäljittelemään malleja, jotka perustuvat ihmisten henkilökohtaisiin

kokemuksiin tai alitajuisiin ajatusmalleihin. Yhteistyötä ihmisen kanssa tarvitaan myös silloin, kun toimintaympäristö on kovin epävarma tai monimutkainen. Tekoälyn käyttöön päätöksenteossa liittyy myös riskejä. Riskit liittyvät erityisesti tapauksiin, joissa tulkintaa vaativat päätökset delegoidaan kokonaan tekoälyjärjestelmälle [15]. Tällöin ihmisen on mahdoton valvoa järjestelmiä, ehkäistä tai korjata syntyviä virheitä tai niiden aiheuttamia haittoja.

On tärkeää esittää kysymyksiä, kuten voiko datasta opittu malli antaa meille oikeiden johtopäätösten lisäksi myös selityksen taustalla oleville ilmiöille. Kun pohditaan esimerkiksi tekoälyn puolueellisuutta, voimmeko olla varmoja siitä, ettei tekoälyjärjestelmä ole poiminut harjoitteludatan puutteista tai virheistä johtuvaa puolueellista kuvaa maailmasta. Ohjelmiston tilaajalle on siis varmistettava riittävä luottamus siitä, että tekoälyjärjestelmän tekemät päätökset on tehty oikeudenmukaisesti ja oikeiden tietojen pohjalta [21]. Adadin ja Berradan mukaan selitettävyys ei ole tärkeää vain päätösten perustelemiseksi, vaan se voi myös estää virheiden syntymistä. Mitä paremmin käyttäjä ymmärtää järjestelmän toimintaa, sitä paremmin hän voi tunnistaa haavoittuvuuksia ja puutteita [14].

3.3 Yhteenveto kirjallisuuskatsauksen tuloksista

Tässä kappaleessa vedetään yhteen kirjallisuuskatsauksen tärkeimmät tulokset. Näitä tuloksia hyödynnetään ja sovelletaan empiirisessä tutkimuksessa.

Vähäkoodinen ohjelmistokehitys (low-code) tarkoittaa ohjelmistojen kehittämistä vähäkoodisilla kehitysalustoilla (low-code development platforms) [43]. Vähäkoodinen ohjelmistokehitys on ohjelmistojen kehittämistä visuaalisten Vedä ja pudota –komponenttien avulla [2]. Tyypillisiä vähäkoodisilla kehitysalustoilla toteutettuja ohjelmistoja ovat muun muassa mobiilisovellukset, sovellukset manuaalisen työn vähentämiseksi ja liiketoimintaa tukevat sovellukset [34]. Tero Tapanainen pohtii eCraftin blogissa, että vähäkoodinen ohjelmistokehitys sopii parhaiten sellaiseen ohjelmistokehitykseen, jossa ohjelmiston ominaisuuksien määrä on suhteellisen rajattu [13]. Tapanaisen mukaan vähäkoodisesta ohjelmistokehityksestä hyötyy eniten, kun ohjelmiston monimutkaisuusaste ei ole suuri, mutta tekemisen volyymiä on paljon.

Monet IT-toimittajat lupaavat, että vähäkoodinen ohjelmistokehitys sopii sekä yksittäisten liiketoimintaa tehostavien ohjelmistojen että laajempien kokonaisratkaisujen toteuttamiseen. On kuitenkin selvää, ettei vähäkoodinen ohjelmistokehitys korvaa kokonaan perinteistä sovelluskehitystyötä eikä se sovellu kaikkeen ohjelmistokehitykseen. Sitä ei kannata käyttää sellaisten

laajojen ja monimutkaisten järjestelmien kehittämiseen, joita varten markkinoilla on olemassa valmiit ohjelmistoratkaisut. Vaativampi koodaustyö, kuten esimerkiksi olemassaolevien toisiinsa kytkettyjen ja omaa räätälöintiä sisältävien järjestelmien kehittäminen, ei kuulu vähäkoodisen ohjelmistokehityksen ensisijaisiin käyttökohteisiin. Vähäkoodisuutta on helpointa toteuttaa uusissa sovelluksissa, jotka tuovat ja vievät suhteellisen yksinkertaista tietoa muihin järjestelmiin.

Vähäkoodisessa ohjelmistokehityksessä tilaajan liiketoiminnan asiantuntijat voivat olla avainroolissa, vaikka heiltä puuttuukin ohjelmointiosaamista [2]. Asiantuntijat tuovat kehittämiseen mukaan prosessiosaamista, he tuntevat työnsä pullonkaulat ja heillä voi olla erinomaisia ehdotuksia niiden ratkaisemiseksi [27]. Kansalaiskehittäjien mukaan ottaminen sisältää valtavasti mahdollisuuksia, mutta siinä nähdään myös laatu- ja turvallisuusriskejä. Kansalaiskehittäjiltä vaaditaan hyvää teknistä ja liiketoiminnallista ymmärrystä, jotta he yksinään pystyisivät kehittämään toimivia ohjelmistoja. Yleinen käsitys on, että vähäkoodisilla kehitysalustoilla ohjelmistojen kehittämiseen kannattaa ottaa IT-kumppani mukaan [27]. Kehittämisprojekteihin tarvitaan myös yrityksen IT-asiantuntijoita [49]. Varsinkin jos ohjelmisto sisältää monimutkaisia rajapintoja muihin järjestelmiin, mukaan tarvitaan yrityksen sovellusarkkitehtuuria tuntevia henkilöitä.

EU haluaa tekoälyyn liittyvällä asetusehdotuksella varmistaa, että ohjelmistojen käyttäjät voivat luottaa tekoälyyn ja sen käyttöön mahdollisimman laajasti [44]. Asetusehdotuksessa tekoälyjärjestelmät luokitellaan neljään riskiryhmään käyttötarkoituksen pohjalta. Asetusehdotus ottaa eniten kantaa korkeariskisten järjestelmien vaatimuksiin. Vähäriskisten tekoälyjärjestelmien kohdalla on riittävää, että käyttäjä tietää olevansa vuorovaikutuksessa tekoälyn kanssa. Asetusehdotuksessa huomioidaan myös muun muassa sisältöjen manipulointi, josta tulee ilmoittaa käyttäjälle. Käyttäjälle tulee ilmoittaa, jos esimerkiksi henkilö, esine tai paikka voi vaikuttaa aidolta ja käyttäjä voi erehtyä luulemaan sitä todelliseksi.

Markkinoilla on useita vähäkoodisia ohjelmistokehitysalustoja, joilla voidaan tuottaa tekoälyä sisältäviä ohjelmistoja. Koska vähäkoodisen kehitysalustan avulla ohjelmistoja voidaan kehittää käyttäen graafisia malleja, erityisesti kansalaiskehittäjän voi olla helpompi ymmärtää mistä ohjelmisto rakentuu ja miten se toimii. Vastuu tekoälyjärjestelmän lopputuloksista on sekä kehittäjillä että niiden käyttäjillä. Myös käyttäjän on ymmärrettävä, mitä tekoäly tekee ja mitä käyttäjältä itseltään vaaditaan [30]. Roolien selkeys vaikuttaa tekoälyn läpinäkyvyyteen ja edistää siten käyttäjien luottamusta tekoälyyn [30].

Tekoälyn ymmärrettävyys paranee, kun ihminen pystyy ymmärtämään ohjelmiston toiminnallisuuksia, siinä käytettävää dataa ja sitä, millaisia rajapintoja ohjelmistolla on ulkoisiin tietoihin ja palveluihin. Selitettävyyttä lisää se, jos ihminen ymmärtää miten tekoälyn päätöksentekomalli on luotu ja miten sitä on koulutettu [6]. Näin käyttäjä voi ymmärtää paremmin sen, mihin tekoälyn tekemät päätökset perustuvat. Black-box malleissa tekoälyn päätöksentekomalli ei avaudu käyttäjälle, jolloin luottamus tekoälyyn voi vähetä [30].

Kun tekoälyn selitettävyyttä lisätään, ihminen voi ymmärtää miksi tekoälyjärjestelmä päätyi tiettyyn lopputulokseen. Ihminen voi ymmärtää järjestelmän toimintaa ja havaita, milloin jokin toiminto onnistuu tai epäonnistuu. Selittävyyden avulla ihmisen on helpompi tietää, milloin järjestelmän tekemiin tuloksiin voi luottaa. EU pyrkii ohjeillaan edistämään luotettavaa tekoälyä [14]. Selitettävyys on keskeinen asia käyttäjien tekoälyjärjestelmiä kohtaan tunteman luottamuksen luomiseksi ja ylläpitämiseksi. Prosessien tulee olla avoimia, tekoälyjärjestelmien tarkoitus on ilmoitettava avoimesti ja päätökset on pystyttävä selittämään niille, joihin ne suoraan tai välillisesti vaikuttavat [14]. Myös Pieters nostaa tutkimuksessaan esiin läpinäkyvyyden ja avoimuuden merkityksen luottamuksen lisäämiseksi [30]. Tekoälyn avoimuus ja läpinäkyvyys liittyvät käyttäjien tarpeeseen ymmärtää tekoälyn toimintaa paremmin [30].

Päätöksenteko on usein yhdistelmä tietojen analysointia ja intuitiota. Tekoälyjärjestelmät mahdollistavat parempien päätösten tekemisen, kun tekoäly voi seuloa valtavia määriä dataa joiden pohjalta ihmiset voivat intuition avulla tehdä päätöksiä [26]. Ashoorin ja Weiszin tutkimuksessa ihmisen osallistumista tekoälyn päätöksentekoon pidettiin oleellisena luottamuksen kannalta. Osallistujien mielestä oli tärkeää, että ihminen on mukana joko varsinaisessa päätöksenteossa tai vähintään tekoälyn kouluttajana [6]. Tekoäly voi täydentää ihmisen päätöksentekoa, mutta se ei monessakaan tapauksessa korvaa sitä.

Luottamus teknologiaan ja tekoälyyn voidaan määritellä hyödyntämällä Coeckelberghin määritelmää, jonka mukaan luottamus tekoälyyn tarkoittaa oletusta siitä, että sen odotetaan toimivan sille tarkoitetulla tavalla ihmisen asettaman tavoitteen saavuttamiseksi [9].

4 Empiirinen tutkimus

Empiirisen tutkimuksen tutkimusmenetelmäksi valitsin konstruktiivisen tutkimuksen, koska tutkimuksen tavoitteena oli tuottaa konkreettinen lopputulos eli sovellus. Empiirinen tutkimus toteutettiin tapaustutkimuksen (case study) sekä puolistrukturoitujen haastattelujen avulla.

Tässä luvussa kuvataan empiirisen tutkimuksen tulokset. Aluksi esitellään olemassa olevan rekrytointipelin toiminnallisuudet ja sen jälkeen kerrotaan, miten työpaikkojen suosittelu –sovellus suunniteltiin, toteutettiin ja käyttöönotettiin vähäkoodisella OutSystems-kehitysalustalla. Tapaustutkimuksen tuloksissa pyritään vastaamaan tutkimuskysymykseen "miten vähäkoodisella ohjelmistokehityksellä luodaan tekoälyä hyödyntäviä ohjelmistoja". Lopuksi kuvataan haastattelujen tulokset ja vastataan tutkimuskysymykseen "miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta".

4.1 Case study: työpaikkojen suosittelu -sovellus

Tapaustutkimuksen tavoitteena oli suunnitella ja toteuttaa yksinkertainen tekoälyä hyödyntävä työpaikkojen suosittelu -sovellus, jonka kehitysalustana toimii OutSystems Platform. Tapaustutkimuksessa käytettiin pohjana olemassa olevaa rekrytointipeliä. Pelissä pelaaja ratkaisee koodaustehtäviä neljällä eri ohjelmointikielellä. Ohjelmointitehtävien jälkeen pelaajan on mahdollista siirtyä suosittelu-osioon (myöhemmin sovellus), jossa pelaaja saa suosituksia siitä, mihin Solitan avoimeen työtehtävään hänen tiedoillaan ja osaamisellaan kannattaisi hakea. Suosittelu-osio perustuu sekä pelaajan ratkaisemiin ohjelmointitehtäviin että hänen antamiinsa taustatietoihin. Taustatiedot koostuvat työkokemukseen ja opiskelutaustaan liittyvistä kysymyksistä. Suosittelu-osio perustuu tekoälyyn, joka kehitetään osana tätä tutkimusta. Rekrytointipeli on Solitan aikaisemmin tuottama peli, jonka toiminnallisuuksia ei tässä tutkimuksessa ole muutettu. Varsinaisen pelin sisältö, ulkoasu ja toiminnallisuudet on rajattu tämän tutkimuksen ulkopuolelle.

Suosittelu-osion tilaajana tässä tutkimuksessa pidetään Solitan HR-yksikköä. HR-yksikkö syöttää sovellukseen Solitan avoimet työpaikat ja osaamisvaatimukset, joita tietyssä työtehtävässä tarvitaan. Sovelluksen tilaaja määritellään siksi, että diplomityön tutkimuskysymyksessä tekoälyn ymmärrettävyyttä ja selitettävyyttä tarkastellaan erityisesti tilaajan näkökulmasta.

Suosittelu-osio kehitettiin alla esitetyn prosessin mukaisesti:

1. Sovelluksen toimintojen kuvaaminen tilaajan kanssa
2. Sovelluksen käyttäjäroolien, käyttötapausten ja sovelluksen käyttämien tietojen määrittely tilaajan vaatimusten perusteella
3. Kehitystyö OutSystems-kehitysalustalla
    - Työkokemuksen, koulutustaustan ja avoimien työpaikkojen tietomallien sekä tietokantarakenteiden kehittäminen
    - Tekoälyä hyödyntävän suosittelu-osion kehittäminen, jossa hyödynnettiin OutSystemsin sisäänrakennettuja komponentteja
4. Kehitettyjen toiminnallisuuksien testaaminen
5. Sovelluksen käyttöönotto

4.1.1 Kuvaus pelin olemassa olevista toiminnallisuuksista

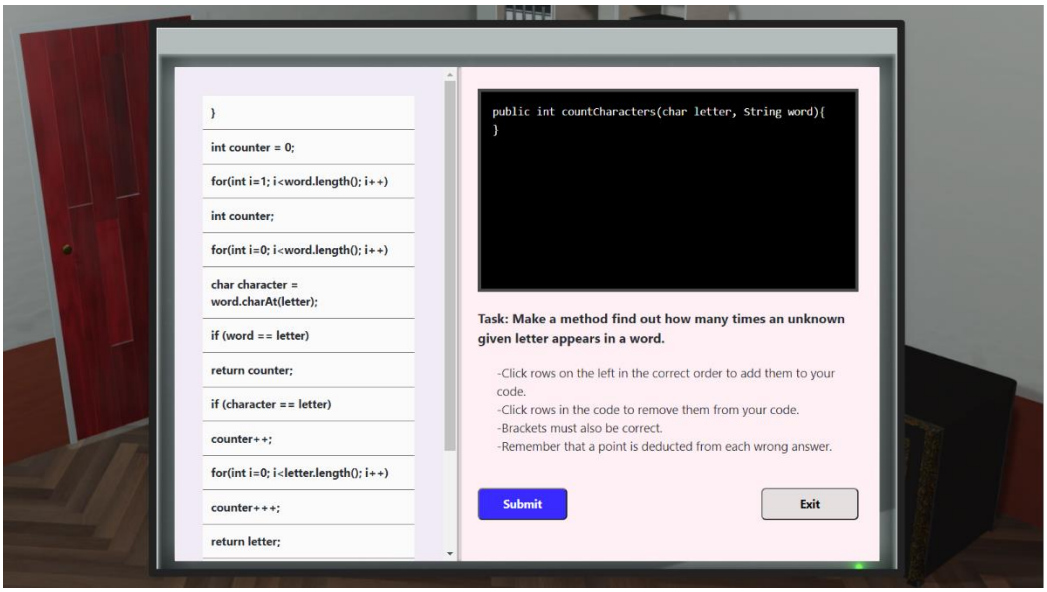
Vaikka rekrytointipelin käyttöliittymää eikä toiminnallisuuksia ole muutettu tässä tutkimuksessa, niiden ymmärtäminen tukee suosittelu-osion logiikan ymmärtämistä. Siksi toiminnallisuudet kuvataan tässä kappaleessa. Kuvissa 10-13 on kuvakaappauksia rekrytointipelin näytöistä.

Pelin ideana on pakohuonepelimäinen tilanne, jossa pelaaja siirtyy aloitushuoneen kautta seuraaviin huoneisiin. Huoneita on neljä ja jokainen niistä sisältää yhden ohjelmointikielen tehtäviä (Python, Java Script, Java, SQL). Aloitushuoneesta pelaaja voi vapaasti siirtyä mihin tahansa huoneeseen. Huoneeseen mentyään hänen on ratkaistava eri tasoisia ohjelmointitehtäviä edetäkseen tehtävästä toiseen ja huoneesta toiseen. Oikein vastaaminen tarjoaa pelaajalle seuraavan vinkin siitä, kuinka hän pääsee seuraavaan huoneeseen. Pelaaja menettää pisteitään, mikäli hän kokeilee tehtäviä monta kertaa väärillä arvoilla. Tällöin pelaaja ei saa tehtävästä täysiä pisteitä, vaikka hän ratkaisisi tehtävän oikein. Kun pelaaja on ratkaissut joko kaikki tai haluamansa määrän tehtäviä, hän voi poistua pelistä Excape-huoneen kautta. Tällöin pelaaja näkee pisteensä sekä kustakin huoneesta erikseen että yhteenvedon kaikista huoneista.

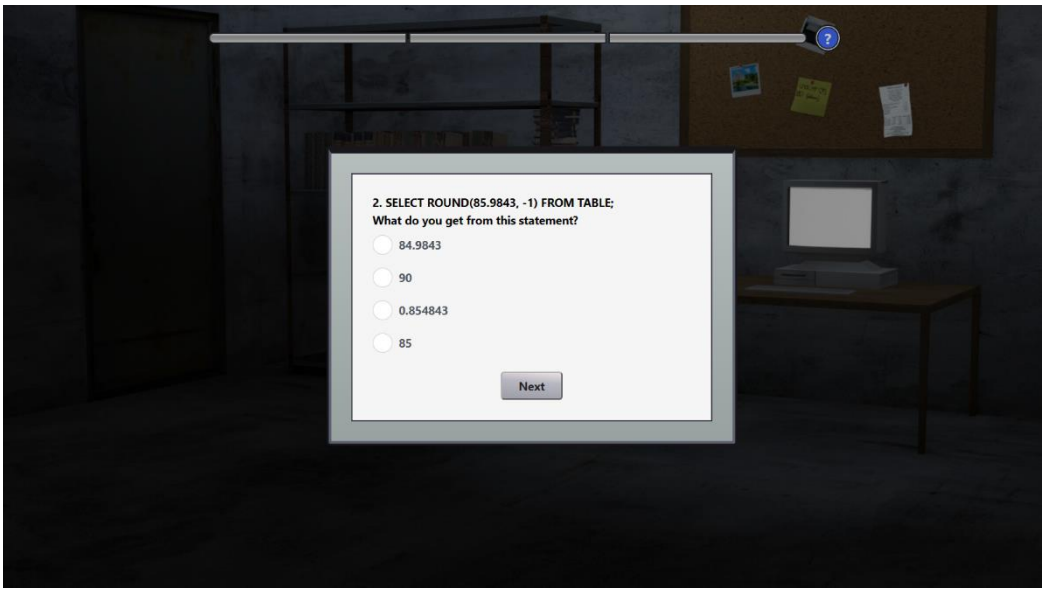
Kuva 10: Rekrytointipelin aloitussivu, aloitussivulla pelaaja valitsee minkä ohjelmointikielen tehtäviä hän haluaa ratkaista.

Pelaaja voi vapaasti valita haluamansa huoneen eli ohjelmointikielen. Hän pystyy myös vaihtamaan huonetta, mikäli hän ei pysty ratkaisemaan tehtäviä. Pelaajan ei siis tarvitse suorittaa jokaista ohjelmointikielen polkua loppuun päästäkseen seuraavaan. Tässä tutkimuksessa toteutettavan lisäosan kannalta on hyödyllistä, että peli laskee pisteet erikseen jokaisesta huoneesta. Pelaajan osaaminen voidaan näin määrittää ohjelmointikielittäin. Peli tarjoaa pelaajalle jokaisesta huoneesta maksimissaan 100 pistettä eli yhteensä 400 pistettä, mikäli pelaaja ratkaisee kaikki tehtävät oikein loppuun asti.

Kuva 11: Java huoneen aloitus.



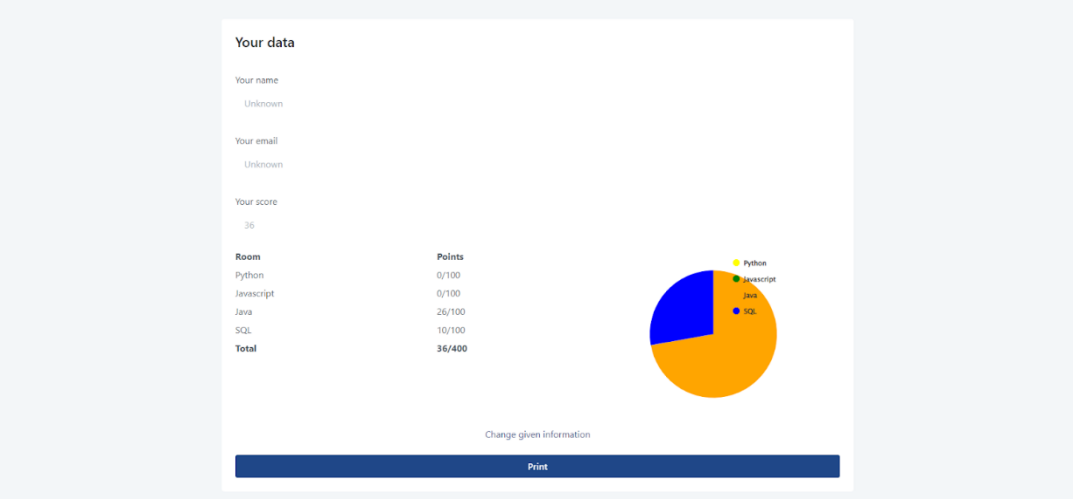
Kuva 12: Esimerkki Java huoneen ohjelmointitehtävästä.



Kuva 13: Esimerkki SQL huoneen ensimmäisestä tehtävästä.

Pelin loputtua käyttäjältä kysytään nimeä ja sähköpostiosoitetta. Tietojen antaminen ei ole pakollista, vaan pelaaja voi nähdä saamansa pisteet myös antamatta tietojaan. Tulossivulla pelaajalle näytetään sekä yhteispisteet että graafinen jakauma siitä, kuinka pelaaja onnistui eri ohjelmointikielien tehtävissä. Lopuksi pelaaja voi halutessaan tulostaa tulokset ja lähettää ne omaan sähköpostiosoitteeseensa. Tästä on kuvakaappaus kuvassa 14.

Pelin tuloksia ei aiemmin hyödynnetty lainkaan, paitsi siten että pelaaja sai jonkinlaisen käsityksen oman osaamisensa tasosta. Tuloksia ei voitu hyödyntää Solitalla, koska tulokset eivät tallentuneet mihinkään. Diplomityön aiheen selvittyä keskustelimme Solitan HR:n kanssa pelin kehittämistoiveista ja siitä, voisiko peliä laajentaa suosittelu-osiolla. Suosittelu-osion tavoitteena on hyödyntää pelin tuloksia ja tuottaa lisäarvoa sekä pelaajalla että Solitalle.



Kuva 14: Rekrytointipelin lopetussivu, jossa pelaajalle näytetään pisteiden lisäksi graafinen kuvaaja pisteiden jakautumisesta.

4.1.2 Sovelluksen suunnittelu

Sovelluksen suunnittelu aloitettiin kuvaamalla yhteistyössä tilaajan kanssa sovelluksen toiminnot. Tilaajan sovellukselta toivomat ominaisuudet kirjattiin ylös ja niiden pohjalta tehtiin määrittely.

Sovelluksen suunnittelussa huomioitiin EU:n tekoälyasetusehdotuksen vaatimukset. Asetusehdotuksessa tekoälyjärjestelmät luokitellaan neljään riskiryhmään käyttötarkoituksen pohjalta [44]. EU:n mukaan korkean riskin tekoälyjärjestelmiä ovat esimerkiksi järjestelmät, joita käytetään rekrytoinnissa työhakemusten karsimiseen. Vaikka tapaustutkimuksessa toteutettu sovellus ei ole varsinainen rekrytointisovellus, huomioidaan siinä tekoälyasetusehdotuksen kyseinen vaatimus. Pelaajalta ei kysytä taustatietoina sellaisia tietoja, joilla voisi olla vaikutusta henkilöiden yhdenvertaiseen kohteluun. Avoimen työpaikan suositus ei saa kohdistua henkilön sukupuoleen, kansalaisuuteen, ikään tms. Tilaajan, ja myös pelaajan, pitää pystyä ymmärtämään kriteerit, joilla tietyn työpaikan suositus on syntynyt. Yhdenvertaisuussäännöksen mukaan kiellettyjä erotteluperusteita ovat sukupuoli, ikä, alkuperä, kieli, uskonto, vakaumus, mielipide, terveydentila ja vammaisuus [44].

Sovelluksen toiminnot

Sovellus kysyy nimen ja sähköpostiosoitteen lisäksi pelaajan työkokemusta sekä opintotietoja. Pelin loputtua pelaaja siirtyy sivulle, jossa näytetään pelistä saadut pisteet sekä työpaikkasuositus. Sivulta löytyy myös kyseisten työpaikkojen rekrytoinnista vastaavan henkilön yhteystiedot ja linkki Solitan rekrytointisivustolle.

Sovellukseen tallennetaan Solitan avoimet ja lähiaikoina avautuvat työpaikat. Avoimille työpaikoille luodaan pistevaatimukset, joiden tulee täyttyä, jotta peli suosittelee pelaajalle kyseistä työpaikkaa. Jokaiselle ohjelmointikielelle annetaan tavoitepisteet, samoin tavoitepisteet työkokemukselle ja koulutustaustalle. Kullekin työpaikalle voidaan työkokemuksen ja koulutustaustan kohdalle määrittää 1-3 vaihtoehtoa, joista vähintään yhden pitää täyttyä, jotta sovellus suosittelee pelaajalle kyseistä työpaikkaa. Näille vaihtoehdoille määritetään tärkeysjärjestys. Jokaiseen kohtaan (ohjelmointikieleen, työkokemukseen ja opiskelutaustaan) voidaan jokaisen työpaikan kohdalla laittaa myös tieto, ettei pistemäärällä ole väliä.

Pisteiden määrittelyn jälkeen tekoäly suosittelee käyttäjän saamien pisteiden sekä työpaikkojen vaatimusten perusteella hänelle sopivimmat työpaikat Solitalla. Sovellus näyttää työpaikan ja siihen liittyvän soveltuvuusprosentin, joka määräytyy käyttäjän pelistä saatujen pisteiden, työkokemuksen ja koulutustaustan perusteella. Avointen työpaikkojen ehdotukset esitetään prosenttijärjestyksessä, eli ensimmäiseksi tulee työpaikka, jonka vaatimukset käyttäjä on täyttänyt parhaiten.

Käyttötapaukset

Käyttäjärooleja tunnistettiin kaksi: pelaaja ja HR. Lisäksi tunnistettiin käyttötapaukset, joilla kuvattiin käyttäjien sovelluksessa suorittamia tehtäviä.

Pelaajan käyttötapaukset:

- Pelaaja lopettaa pelin ja saa pisteet suoritettujen tehtävien perusteella (vanha käyttötapaus)
- Pelaaja antaa nimen ja sähköpostiosoitteen (vanha käyttötapaus)
- Pelaaja valitsee sopivan koulutustaustatiedon
- Pelaaja valitsee sopivan työkokemustiedon
- Pelaaja pyytää suosituksia antamiensa taustatietojen ja saatujen pisteiden perusteella
- Pelaaja näkee suositeltavat työpaikat

- Pelaaja näkee kyseisten työpaikkojen rekrytoinnista vastaavan henkilön yhteystiedot ja linkin Solitan rekrytointisivustolle
- Pelaaja voi lähettää tiedot omaan sähköpostiosoitteeseensa

HR:n käyttötapaukset:

- HR tallentaa avoimet työpaikat ja niiden koulutustausta- ja työkokemusvaatimukset sekä rekrytoinnista vastaavan henkilön yhteystiedot

Tiedot

Sovelluksen kuvauksen perusteella tunnistettiin sovelluksen käyttämät tiedot yleisellä tasolla. Tiedot listattiin taulukkoon 5.

Taulukko 5: Sovelluksen tiedot.

Tieto 1	**Tieto 2**	**Tieto 3**	**Huomiot**
Pelaaja	Nimi		
	Sähköpostiosoite		
	Pelaajan kokonaispisteet	Pelistä saadut pisteet	Python, Java Script, Java, SQL
		Koulutus	
		Työkokemus	
Avoin työpaikka	Nimi		
	Kuvaus		
	Rekrytoinnista vastaavan henkilö	Nimi Yhteystiedot	
	Vaaditut pisteet	Pisteet ohjelmointikielittäin	Python, Java Script, Java, SQL
		Koulutustausta-vaatimus	
		Työkokemus-vaatimus	
Suositeltava työpaikka			Lopputulos, jossa verrataan pelaajan kokonaispisteitä työpaikan pisteisiin

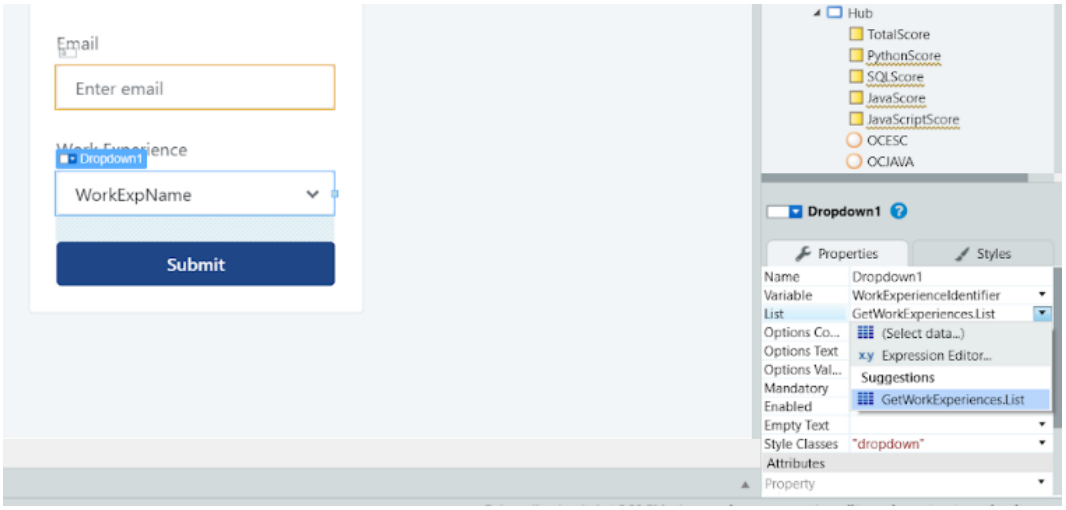
4.1.3 Sovelluksen toteutus

Sovellus toteutettiin OutSystemsin kehitysalustalla. OutSystemsilla toteutetut sovellukset rakennetaan kehitysalustan Service Studio -osiossa, joka on visuaalinen vähäkoodinen ohjelmointiympäristö. Service Studio koostuu eri toiminnallisuuksia sisältävistä välilehdistä, joita ovat käyttöliittymä, logiikka, data ja prosessit [25].

Tietojen keruu

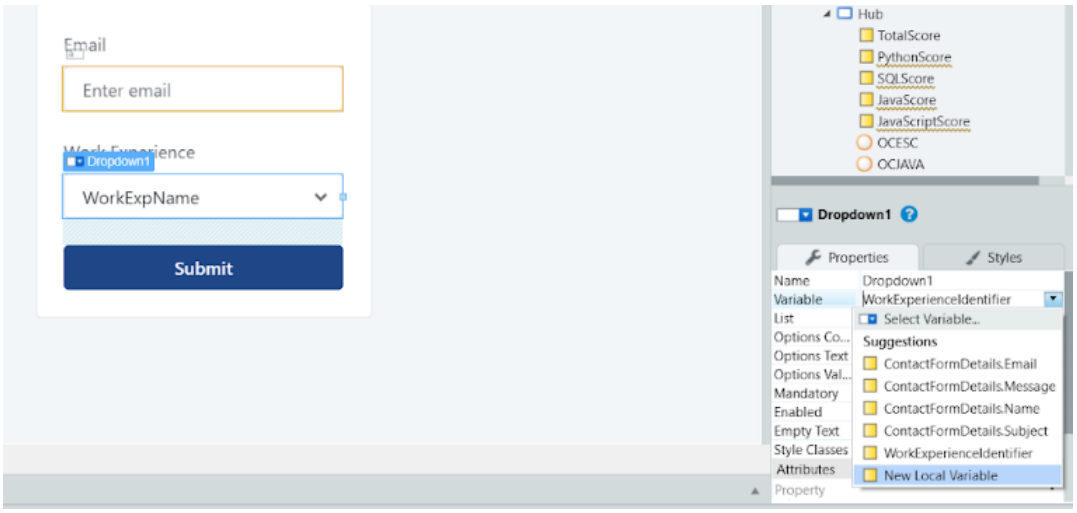
Ensimmäisenä toiminnallisuutena toteutettiin työkokemukseen ja koulutustaustaan liittyvien tietojen keruu. Tiedot lisättiin olemassa olevalle sivulle, jolla rekrytointipelissä kysyttiin nimeä ja sähköpostiosoitetta. Pelaajalta kerättyjen tietojen keruun OutSystems-toimintojen kuvakaappaukset ovat kuvissa 17-20.

Työkokemus ja koulutustausta toteutettiin vetovalikkoina, sillä pelaajalle haluttiin antaa koodistosta valmiit vaihtoehdot. Toteutus aloitettiin työkokemuksesta. Työkokemukselle luotiin tietokantaan entiteetti WorkExperience, jolle määritettiin attribuutit WorkExpName ja WorkExpIndex. WorkExpName attribuutti pitää sisällään työkokemuksen nimen, esimerkiksi "No experience" ja "IT sector 1-2 years". WorkExpIndexin sisältämä numero määrittelee työkokemusten järjestyksen (tilaajan määrittelemä tärkeysjärjestys). OutSystems luo automaattisesti jokaiselle entiteetille identifierin, jolla tiedot yksilöidään. Myös WorkExperience entiteetille luotiin identifier, jota hyödynnetään suositusten antamisessa.



Kuva 17: Tietokannan valinta työkokemuksen vetolaatikon arvoksi.

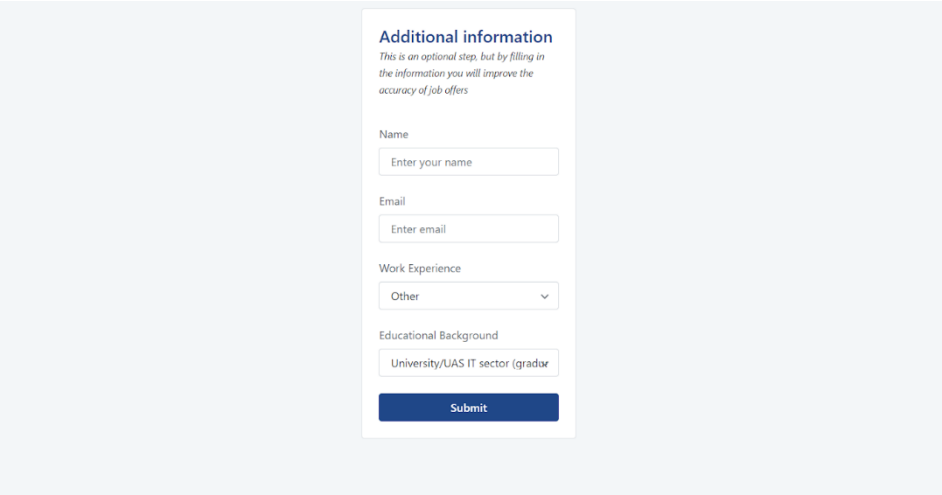
Tietokannan entiteetin luonnin jälkeen toteutuksessa siirryttiin Interface sivulle ja tuotiin kyseisen tietokannan tiedot halutulle (Form) Screenille käyttämällä OutSystemsin sisäänrakennettua "Fetch Data from Database" toimintoa. Tämän jälkeen Form Screenille pystyttiin luomaan vetovalikko OutSystemsin valmiiden vedä ja pudota -komponenttien avulla. Kyseiselle vetovalikolle tarvitsi tämän jälkeen määrittää enää lista, josta komponentti hakee tietonsa sekä luoda local variable tiedon tallentamista varten. OutSystems osaa suositella käyttäjälle oikeaa listaa kyseiselle Screenille tuodun entiteetin avulla. Myös local variablen luominen onnistuu yhdellä painalluksella OutSystemsin tarjoaman suosituksen avulla.



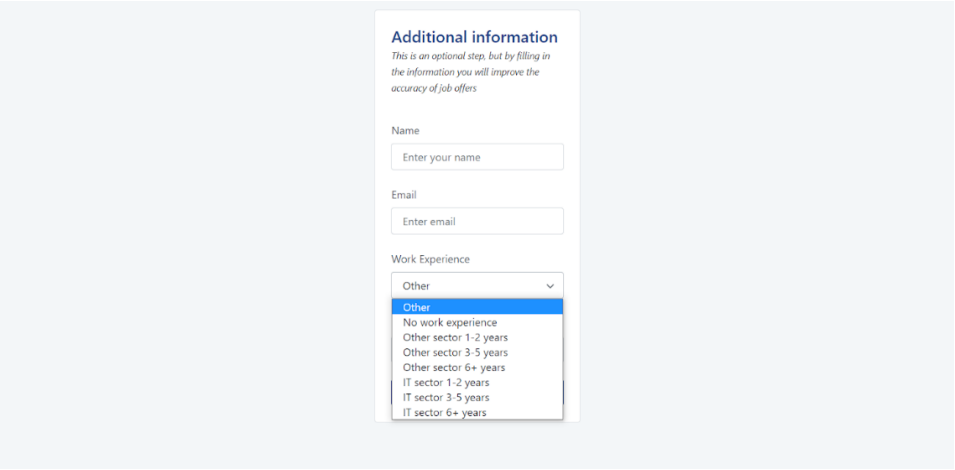
Kuva 18: Valinta muuttujasta, johon käyttäjän työkokemus tallennetaan.

Dropdownin tiedolle luotiin tallennusmahdollisuus luomalla uusi lokaali muuttuja yhdellä napautukselle. OutSystems nimesi muuttujan automaattisesti helposti ymmärrettävällä termillä.

Näiden vaiheiden jälkeen työkokemuksen vetovalikko oli tehtynä ja lokaali muuttuja WorkExperienceIdentifier oli valmis hyödynnettäväksi. Työkokemuksen jälkeen loin myös koulutustaustalle vetovalikon, joka toteutettiin samalla tavalla kuin työkokemuksen vetovalikko. Ainoat muutokset olivat tietokannassa, johon EducationalBackground entiteettiin luotiin EduName attribuutti koulutuksen nimeä varten, sekä EduIndex koulutuksesta saatavia pisteitä varten.



Kuva 19: Tietojen keruu osio, kun vetovalikot työkokemuksesta ja koulutustaustasta on lisätty.



Kuva 20: Tietojen keruu osion vetovalikon tiedot, jotka on tuotu työkokemus tietokannasta.

Avoinna olevien työpaikkojen tiedot

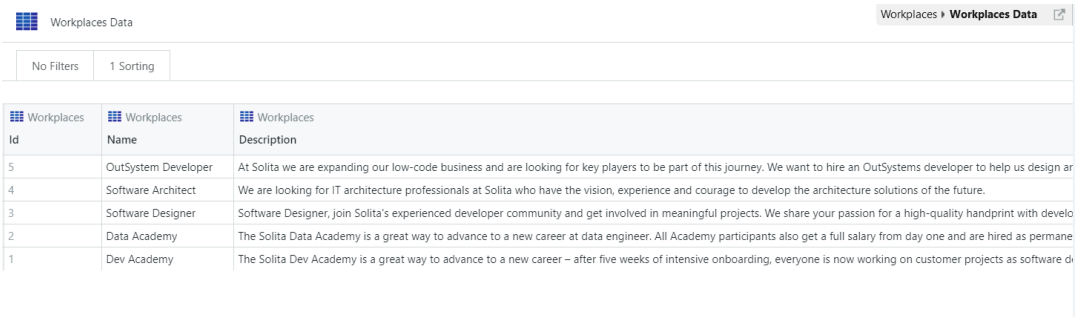
Avoinna olevat tai avoimeksi tulevat työpaikat luodaan käsin järjestelmässä olevaan Workplaces tietokantaan. Tietokanta sisältää 16 attribuuttia, jotka on esitetty taulukossa 6.

Taulukko 6: Tietokannan data ja attribuutit.

Data nimi	Data tyyppi	Selitys
Id	Id	OutSystemsin automaattisesti luoma id.
Name	Text	Työpaikan nimi
Description	Text	Työpaikan kuvaus, joka näytetään käyttäjälle suositusten yhteydessä.
Contact	Text	Työpaikan kontaktihenkilö
ContactEmail	email	Työpaikan kontaktihenkilön sähköposti
WEW1	Text	Työpaikan 1. toive käyttäjän työkokemuksesta
WEW2	Text	Työpaikan 2. toive käyttäjän työkokemuksesta
WEW3	Text	Työpaikan 3. toive käyttäjän työkokemuksesta
EBW1	Text	Työpaikan 1. toive käyttäjän koulutustaustasta
EBW2	Text	Työpaikan 2. toive käyttäjän koulutustaustasta
EBW3	Text	Työpaikan 3. toive käyttäjän koulutustaustasta

MinPython	Integer	Työpaikan vähimmäispistemäärä vaatimus ohjelmointikielessä Python, joka käyttäjän tulee saada. Pisteet 0-100, 0 tarkoittaa ettei ohjelmointikieltä oteta huomioon suosituksia annettaessa.
MinJava	Integer	Työpaikan vähimmäispistemäärä vaatimus ohjelmointikielessä Java, joka käyttäjän tulee saada. Pisteet 0-100, 0 tarkoittaa ettei ohjelmointikieltä oteta huomioon suosituksia annettaessa.
MinJS	Integer	Työpaikan vähimmäispistemäärä vaatimus ohjelmointikielessä JavaScript, joka käyttäjän tulee saada. Pisteet 0-100, 0 tarkoittaa ettei ohjelmointikieltä oteta huomioon suosituksia annettaessa.
MinSQL	Integer	Työpaikan vähimmäispistemäärä vaatimus ohjelmointikielessä SQL, joka käyttäjän tulee saada. Pisteet 0-100, 0 tarkoittaa ettei ohjelmointikieltä oteta huomioon suosituksia annettaessa.
Points	Integer	Käyttäjän saamat pisteet kyseiseen työpaikkaan. Vakioarvo on 0. Numeroa päivitetään, kun käyttäjä siirtyy suositussivulle.

Tietokanta luotiin OutSystemsin valmiilla toiminnallisuuksilla ja sitä pystyy päivittämään suoraan OutSystemsin Service Studiosta. Kuvakaappaus on kuvassa 21.



Kuva 21: Tietokannan luominen.

Tietojen siirto ja näyttö pelaajalle

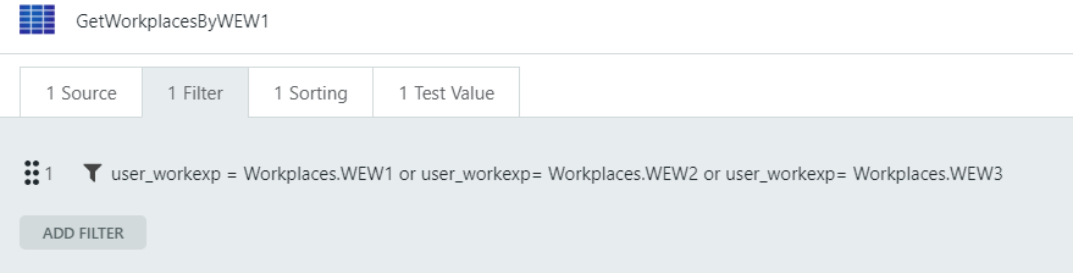
Tiedot pelaajan nimestä, sähköpostista, työkokemuksesta ja koulutustaustasta siirretään OutSystemsin sisällä Report screeniin valmiin Destination komponentin avulla ja tallennetaan Input parametreihin. Report screenillä pelaajalle näytetään hänen syöttämänsä tiedot (nimi, sähköposti, työkokemus ja koulutustausta) sekä hänen pelistä saamansa pisteet kokonaisuudessaan sekä eriteltynä eri ohjelmointikielille. Pelaaja saa myös näkyviin ympyrädiagrammin, josta pelaaja voi nähdä kuinka eri ohjelmointikielistä saadut pisteet jakautuvat.

Pelaaja voi palata pelin loputtua muokkaamaan tietojaan "Change given information" painikkeesta tai siirtyä suositus-sivulle "See recommendations" painikkeesta, jossa hänelle näytetään tekoälyn hänelle suosittelemat työpaikat. Mikäli pelaaja siirtyy suositus-sivulle, OutSystems vie tiedot pelaajan työkokemuksesta ja koulutustaustasta kyseiselle Recommendation screenille valmiin Destination komponentin avulla. Tämän jälkeen kyseiset tiedot tallennetaan input parametreihin. Nimi- ja sähköpostitietoa ei siirretä.

Tekoälyn suositukset

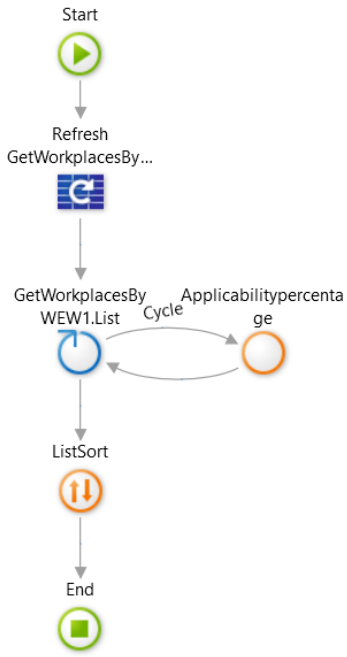
Pelaaja näkee hänelle suositellut avoimet työpaikat, kun hän etenee Report screeniltä Recommendation screenille. Suositukset perustuvat pelaajan antamiin työkokemus- ja koulutustaustatietoihin sekä pelistä saatuihin pisteisiin. Kun pelaaja siirtyy suositussivulle, hänelle aukeaa lista avoimista työpaikoista joita hänelle suositellaan. Lista on OutSystemsin sisäänrakennettu toiminnallisuus, jossa tietokannasta luodaan selkeä UI komponentti.

Kehittäjä voi muokata UI komponentin attribuutteja ja suorittaa listalle erilaisia filteröintejä sekä lajitteluja. Listalle ilmestyvät ne työpaikat, joiden työkokemusehto täyttyy. Ehto on määritelty GetWorkplaces attribuuttiin filterinä, joka tarkastaa, sopiiko käyttäjän antama työkokemus johonkin tietokantaan määritellyistä kolmesta työpaikoista. Mikäli ehto täyttyy, näkyy kyseinen työpaikka listalla. Filtteri on OutSystemsin sisäänrakennettu toiminnallisuus, jossa tietokannan arvot saadaan filtteröityä haluttujen tietojen perusteella. Jotta sain filtterin toimimaan haluamallani tavalla, tuli filtteriin kirjoittaa pieni määrä koodia "user_workexp = Workplaces.WEW1 or user_workexp= Workplaces.WEW2 or user_workexp= Workplaces.WEW3". Tästä on kuvakaappaus kuvassa 22.



Kuva 22: Työpaikkojen näyttäminen perustuu filtteriin, joka tarkistaa onko käyttäjällä työpaikkaan soveltuvaa työkokemusta.

Pelaaja näkee näytöllä myös prosenttiluvun, joka kuvaa hänen soveltuvuuttaan kyseiseen työpaikkaan. Tämä prosenttiluku perustuu tässä tutkimuksessa tarkasteltavaan hyvin yksinkertaistettuun tekoälyyn, joka luo suosituksen sekä suositusprosentin käyttäjän työkokemuksen, koulutustaustan sekä pelistä saatujen pisteiden perusteella.



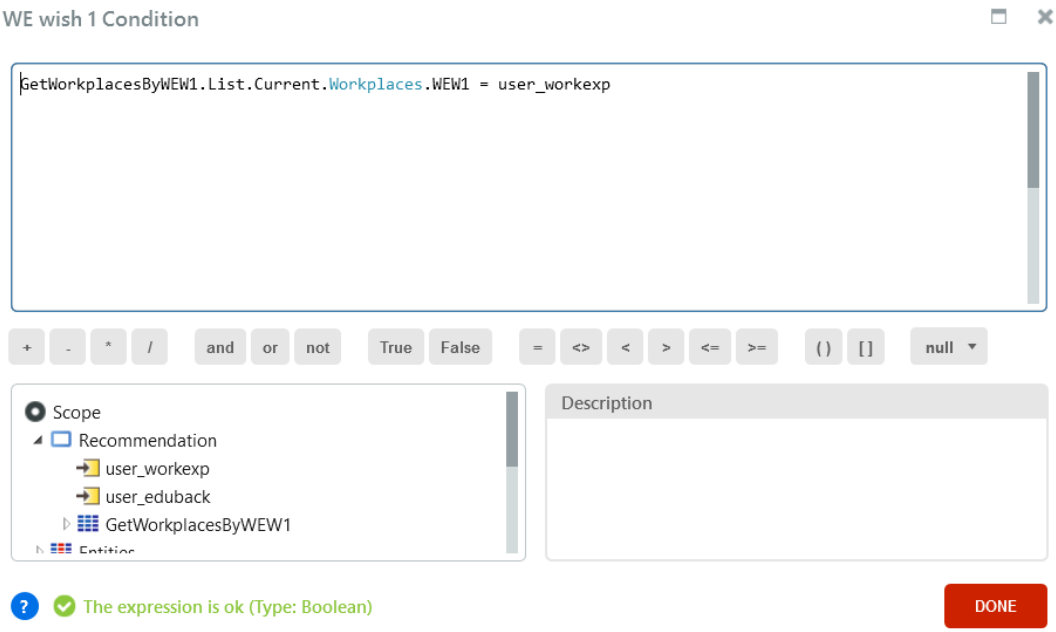
Kuva 23: Logiikka työpaikka suositus -listan läpikäynnin taustalla.

Tekoälyn logiikka on toteutettu ForEach ja Applicabilitypercentage Client actioneihin. Client actionit ovat OutSystemsin sisäänrakennettuja komponentteja, joiden avulla kehittäjä voi toteuttaa erilaisia prosesseja. Client actionit tulee käynnistää sovelluksessa esimerkiksi sivun latauksen yhteydessä, käyttäjän napin painalluksesta tai sivun scrollaamisella alaspäin. Tässä toteutuksessa ForEach client action käynnistetään aina, kun suositussivu latautuu

auki, ja kyseinen ForEach client action käynnistää Applicabilitypercentage client actionin. ForEach client action käy GetWorkplaces listan läpi rivi riviltä ja ajaa jokaiselle listan riville Applicabilitypercentage Client actionin. Listan läpikäymisessä käytettiin Oustystemsin valmista foreach komponenttia, jolle annettiin listaksi filteröity GetWorkplaces lista, jolloin kyseinen komponentti käy läpi jokaisen listan kohdan ja suorittaa sille Applicabilitypercentage client actionin. Näin jokaiseen työpaikkaan saadaan yksilöity prosenttiluku, joka vastaa juuri kyseisen työpaikan vaatimuksia.

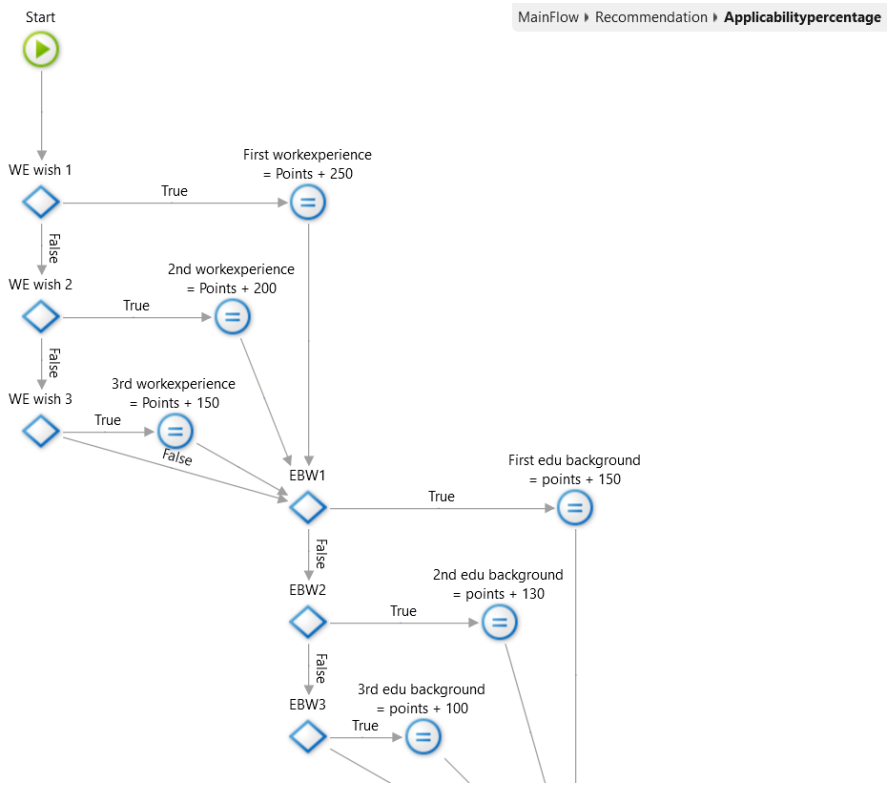
Applicabilitypercentage Client action on toteutettu erilaisin if-else ehdoin sekä näitä seuraavin määrityksin. If-else ehtoja kyseisessä client actionissa on yhteensä 14, kuten myös määrityksiä. Sekä If-komponentit että Assign-komponentit ovat OutSystemsiin sisäänrakennettuja valmiita osia, joita kehittäjä pystyy käyttämään prosesseissaan vetämällä ne prosessikaavioonsa. Tämän jälkeen komponenteille tulee antaa ehdot ja assignments, jotka kirjoitetaan itse tai valitaan OutSystemsin valmiiksi ehdottamista osista. Tässä tutkimuksessa ehdot kirjoitettiin itse hyödyntäen OutSystemsin suosituksia.

Applicabilitypercentage Client action alkaa työkokemuksen tarkastuksella. Action käy läpi kaikki tiettyyn työpaikkaan asetetut työkokemus-vaatimukset ja vertaa niitä käyttäjän antamaan työkokemukseen. Mikäli pelaajan työkokemus vastaa työpaikan ensimmäistä työkokemus-vaatimusta, kyseisen työpaikan tietokantariville lisätään 250 pistettä attribuuttiin Points. Mikäli pelaajan työkokemus vastaa työpaikan toista työkokemus-vaatimusta, käyttäjä saa 200 pistettä ja kolmannesta 150 pistettä. Pisteiden jälkeen logiikassa siirrytään koulutustausta-vaatimusten tarkasteluun. Mikäli pelaajan työkokemus ei vastaa mitään kyseisen työpaikan työkokemus-vaatimusta, sovellus siirtyy suoraan koulutustausta-vaatimuksiin. Pisteitä ei anneta lainkaan. Työkokemusta verrattaessa screeniin tallennettua user_workexp input parametria verrataan Workplaces tietokanta entityn attribuutteihin WEW1, WEW2 ja WEW3. Vertailussa käytetään tekstin vertailua, eli tekstin on oltava kummassakin täsmälleen sama. Kuvassa 24 pelaajan työkokemusta verrataan Workplaces WEW1, WEW2 ja WEW3 attribuutteihin.



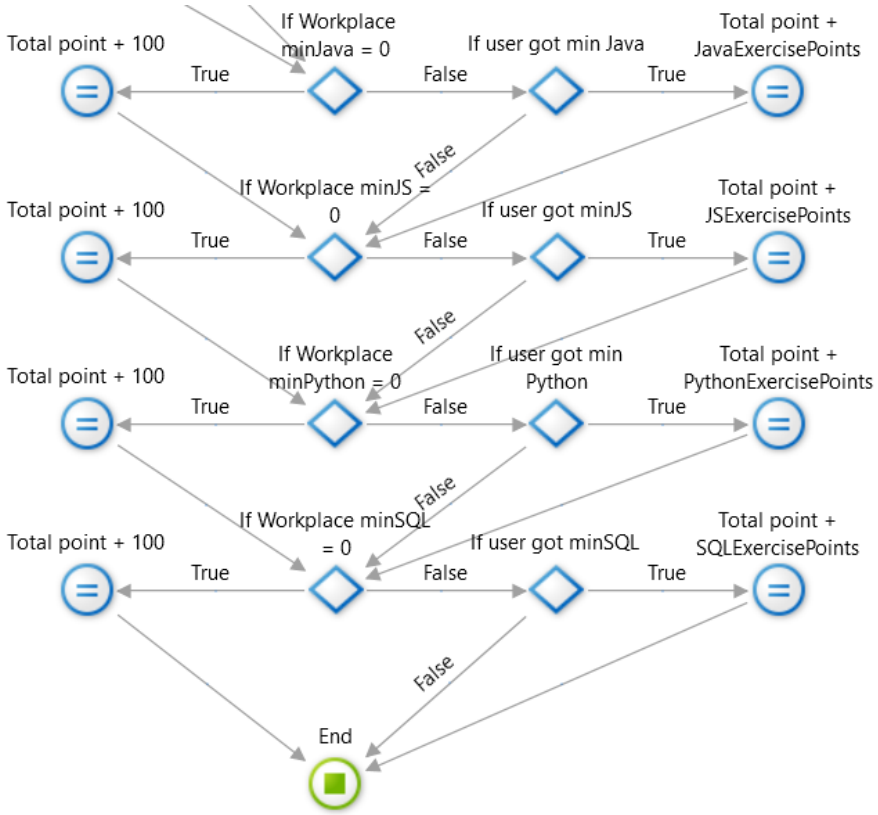
Kuva 24: Pelaajan työkokemusta verrataan Workplaces attribuutteihin.

Koulutustausta-vaatimukset käydään läpi samalla periaatteella kuin työkokemukseen liittyvät vaatimukset. Client action käy läpi työpaikan koulutustausta-vaatimukset järjestyksessä ensimmäisestä kolmanteen, jonka jälkeen pelaajalle (työpaikan tietokantariville Points attribuuttiin) lisätään pisteitä järjestyksen mukaisesti. Mikäli pelaajalla on ensimmäistä koulutustausta-vaatimusta vastaava koulutustausta, hän saa 150 pistettä, toisesta 130 pistettä ja kolmannesta 100 pistettä. Tämän jälkeen siirrytään pelistä saatujen pisteiden ehtoihin. Mikäli pelaajan koulutustausta ei vastaa mikään työpaikan koulutustausta-vaatimusta, siirrytään logiikassa suoraan pelistä saatujen pisteiden ehtoihin. Koulutustaustaa verratessa Recommendation screeniin tallennettua user_eduback input parametria verrataan Workplaces tietokanta entityn attribuutteihin EBW1, EBW2 ja EBW3. Vertailussa käytetään tekstin vertailua, eli tekstin on oltava kummassakin täsmälleen sama.

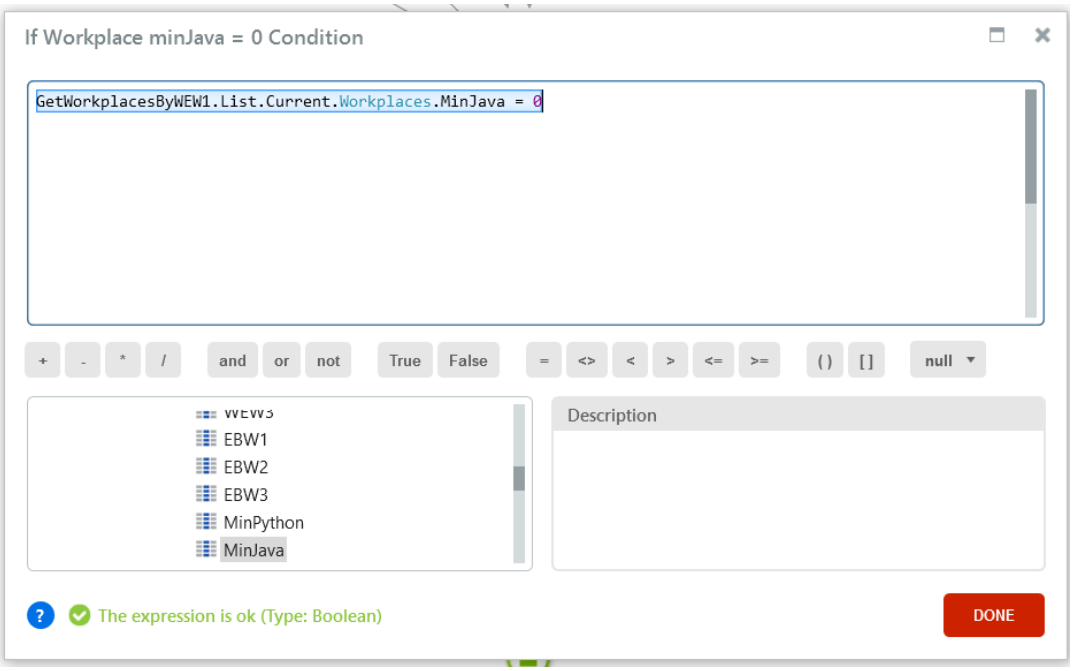


Kuva 25: Applicabilitypercentage client action prosessin työkokemuksen ja koulutustaustan tarkastaminen.

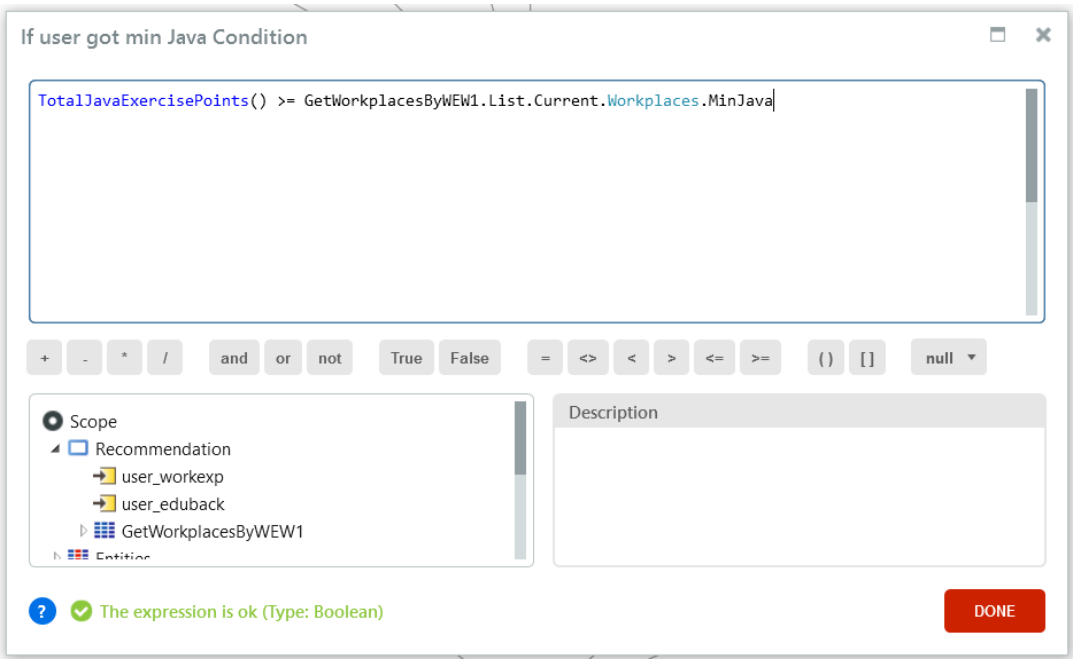
Kun työkokemus ja koulutustausta on käyty läpi, tarkastetaan käyttäjän pelistä saadut pisteet jokaiselle ohjelmointikielelle erikseen. Ensimmäisestä ohjelmointikielestä tarkastetaan sen vaatimus kyseisessä avoimessa työpaikassa. Mikäli työpaikassa kyseinen ohjelmointikieli on asetettu nollaksi (0), ei kielen osaamisella ole työpaikan kannalta väliä ja pelaaja saa automaattisesti kyseisestä ohjelmointikielestä täydet pisteet. Mikäli työpaikassa kyseiselle ohjelmointikielelle on asetettu vaatimuksen alaraja, tarkoittaa se, että kyseinen ohjelmointikieli otetaan mukaan vertailuun. Mikäli pelaajan saamat pisteet ovat yhtä suuret tai suuremmat kuin työpaikan minimivaatimukset kyseiselle ohjelmointikielelle, pelaaja saa pisteitä sen mukaan, kuinka hän on onnistunut ratkaisemaan ohjelmointikielen tehtäviä. Mikäli pelaaja ei saa täytettyä minimivaatimuksia, hän ei saa kyseisestä ohjelmointikielestä yhtään pistettä. Kun ensimmäinen ohjelmointikieli on tarkastettu, siirtyy logiikka seuraavaan ja tarkastaa kaikki neljä ohjelmointikieltä ennen Client actionin lopettamista. Ohjelmointikielien pisteiden tarkastusprosessi on kuvassa 26 ja kuvakaappaukset OutSystemsin näytöistä kuvissa 27-28.



Kuva 26: Prosessi ohjelmointikielien pisteiden tarkastukselle applicabilitypercentage client actionissa.



Kuva 27: Ehdon määrittely, jossa varmistetaan, että työpaikalle on määritetty minimiehto kyseiselle (Java) ohjelmointikielelle.



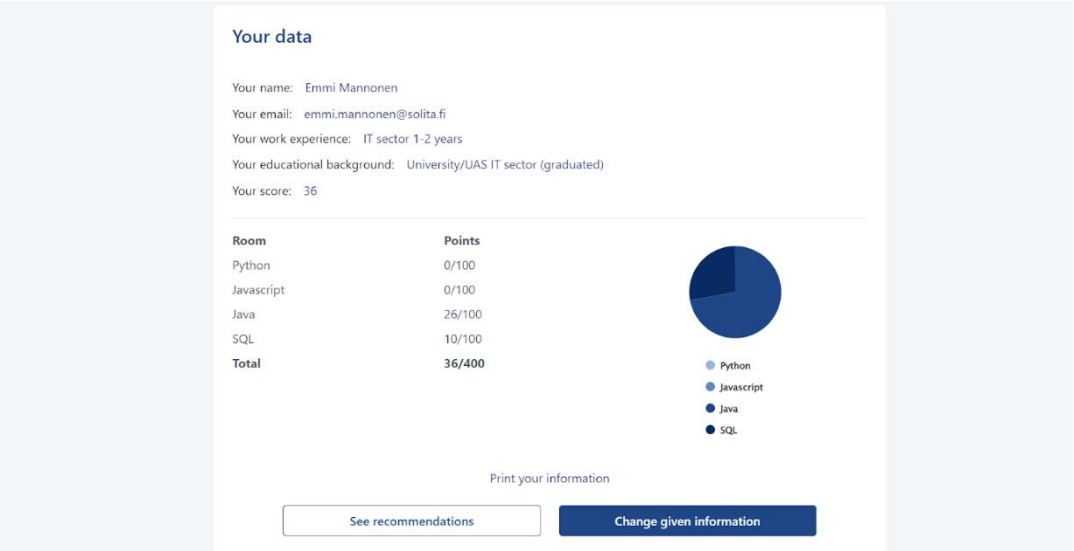
Kuva 28: Ehton määrittely, jossa tarkastetaan, että käyttäjä on saanut kyseisestä koodikielestä (Java) enemmän tai saman verran pisteitä, kun työpaikassa niitä vaaditaan.

Tämän prosessin jälkeen ForEach client action päivittää näytettävän sivun, jossa työpaikat näytetään prosenttilukujen perusteella järjestettyinä paremmuusjärjestykseen. Pelaajalle näytetään hänen saamansa prosenttiluku ja graafi, joka kuvaa kyseistä prosenttilukua. Prosenttiluvun näyttäminen toteutettiin OutSystemsin sisäänrakennetulla Numbers\ProgressCircle komponentilla. Komponenttiin annettiin parametriksi pelaajan työpaikkakohtaiset pisteet jaettuna kuudellasadalla eli täysillä pisteillä.

Pelaajan käyttöliittymät

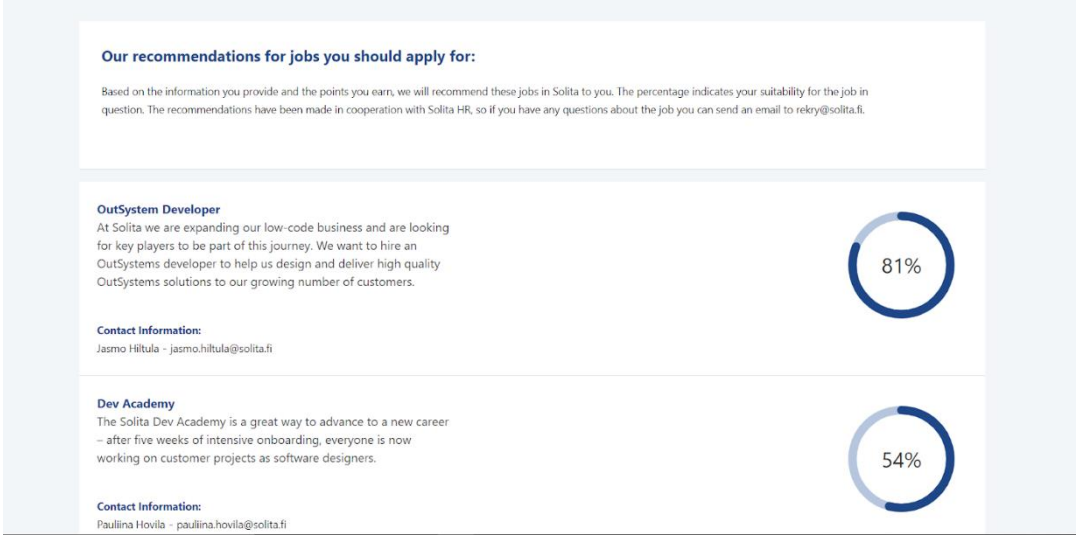
Tilaajan kanssa sovelluksen käyttöliittymäkieleksi valittiin englanti, jotta sovellusta voidaan käyttää kaikissa niissä maissa, joissa Solitalla on toimipisteitä. Käyttöliittymiä ei piirretty etukäteen, sillä niiden suunnittelu ja muokkaaminen on sujuvaa OutSystemsin Service Studion Käyttöliittymä-välilehdellä.

Kuvassa 15 on pelaajan käyttöliittymä, kun pelaajalle näytetään pelaajan antamat taustatiedot, pelistä saadut pisteet ohjelmointikielittäin ja graafinen kuvaaja siitä, miten pisteet jakautuivat eri ohjelmointikielien mukaisesti.



Kuva 15: Sivulla näytetään pelaajan tiedot, pisteet ohjelmointikielittäin ja graafinen kuvaaja pisteiden jakautumisesta.

Kuvan 16 käyttöliittymässä pelaajalle näytetään suositeltavat työpaikat, jotka ovat määräytyneet pelaajan pelistä saatujen pisteiden, työkokemuksen ja koulutustaustan perusteella. Suositeltavat työpaikat esitetään prosenttijärjestyksessä, eli ensimmäiseksi tulee työpaikka, jonka vaatimukset käyttäjä on täyttänyt parhaiten. Sivulta löytyy myös kyseisten työpaikkojen rekrytoinnista vastaavan henkilön yhteystiedot ja linkki Solitan rekrytointisivustolle.



Kuva 16: Sivulla näytetään työpaikat, joita pelaajalle suositellaan haettavaksi.

4.1.4 Sovelluksen käyttöönotto

Sovellukseen tallennettiin ne Solitan työpaikat, joiden haku oli auki tai joihin haku oli avautumassa. Tutkimuksessa työpaikkojen tallennus ja ylläpito tehtiin kehittäjän toimesta, mutta jatkossa avoimet työpaikat voidaan ylläpitää Solitan HR:ssä. Avoimille työpaikoille luotiin pistevaatimukset, joiden tulee täyttyä, jotta peli suosittelee pelaajalle kyseistä työpaikkaa. Jokaiselle ohjelmointikielelle annettiin tavoitepisteet (max 100p/ohjelmointikieli), samoin tavoitepisteet työkokemukselle (max 250p) sekä koulutustaustalle (max 150p). Sovellukseen tallennetuille työpaikoille määritettiin työkokemuksen ja koulutustaustan kohdalle 1-3 vaihtoehtoa, joista vähintään yhden piti täyttyä, jotta sovellus suosittelee pelaajalle kyseistä työpaikkaa. Ohjelmointikieleen, työkokemukseen ja opiskelutaustaan voitiin jokaisen työpaikan kohdalla laittaa myös tieto, jos pistemäärällä ole väliä.

OutSystemsissä sovelluksen käyttöönotto tapahtuu nopeasti ja helposti ns. “1 click publish” -toiminnolla. Toiminto lataa sovelluksen Platform Server –palvelimeen, jonka jälkeen se on tilaajan käytössä.

4.2 Haastattelut

Haastattelujen avulla pyrittiin vastaamaan tutkimuskysymykseen “miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta”. Haastattelukysymykset ovat liitteessä 1. Haastattelevat puhuivat vähäkoodisesta ohjelmistokehityksestä sen englanninkielisellä termillä ”low-code”.

4.2.1 Vähäkoodinen ohjelmistokehitys

Vähäkoodinen ohjelmistokehitys oli kaikille haastateltaville vähintään käsitteen tasolla tuttu. Haastatteluissa toistui yleinen näkemys siitä, että vähäkoodinen ohjelmistokehitys on seuraava iso tekijä ohjelmistokehityksen alalla.

Koen, että low-code on yks next face, yks isompi aalto, joka taas pirstaloi jonkun valmiina olleen systeemin ikään kuin uusiksi. Ei niin että valmiina oleva systeemi poistuu kokonaan, vaan lähestymistapa hommaan muuttuu aika paljon. Vertaan sitä esimerkiksi siihen, että ennen oli HTML joka koodattiin kokonaan omilla käsillä ja sitten tuli kaikki dreamweaver ja muut mitä kautta pystyi teettämään sitä HTML:ää. - HR, 54

Koen, että potentiaalia kyllä löytyy [low-codella] paljon. Saa nähdä miten seuraavina vuosina kehittyy, mutta itse mitä olen sivusta nähnyt niin oon

sitä mieltä että se on varteenotettava kehitysväline. Etenkin nopeus on oman ymmärryksen mukaan isoja tekijöitä. - HR, 33

Näen sellaisena [ohjelmistokehitysvälineenä] joka kasvaa kovasti ja tulee yleistymään ja yleistyykin koko ajan. Sehän näkyy tietysti esim meidän firman [Solitan] sisällä konkreettisesti. - Low-code kehittäjä, 44

Vaikuttaa kyllä siltä, että menee hyvään suuntaan. Aina ei tarvitse keksiä pyörää uudelleen ja siihen koen, että low-code on kyllä hyvä. Just se, että vaikuttaa tehokkaalta kun ei tarvitse käsin koodata samoja asioita monia kertoja. - Ohjelmistokehittäjä 24

Kriittisyys

Vaikka mielipiteet ja käsitykset vähäkoodisesta ohjelmistokehityksestä olivat yleisesti hyvin valoisia ja myönteisiä, suhtautuivat haastateltavat vähäkoodiseen ohjelmistokehitykseen myös varauksella. Etenkin itse vähäkoodisen ohjelmistokehityksen ohjelmistokehittäjät nostivat suurimmaksi haasteeksi vähäkoodisen ohjelmistokehityksen ympärillä olevan kovan hehkutuksen ja hypen. Kriittisyyttä aiheutti myös se, kuinka ja mihin tarkoituksiin vähäkoodista ohjelmistokehitystä käytetään, jotta sen hyödyt pääsevät kunnolla esiin.

Näen sen [low-coden] sellaisena, että siinä on paljon sellaista hypeä, joka menee yli odotuksista mitä siihen liittyy, eli kaikkeen nopeuteen ja sellaiseen liittyen, joka ei kuitenkaan sitten aina toteudu. Mutta kyllä siinä kuitenkin on niitä etuja, että voi olla nopeempaa etenkin jos ratkaisut ovat suhteellisen yksinkertaisia, niin sitten voi olla hyvinkin nopeaa. - Low-code kehittäjä, 44

En missään tapauksessa ajattele, että low-code välineet olis semmonen yleinen työkalu, jota voidaan hyödyntää kaikkeen. - Low-code kehittäjä, 27

Konkreettisia käyttötarkoituksia

Vähäkoodisen ohjelmitokehityksen ohjelmistokehittäjät toivat esiin myös konkreettisia esimerkkejä siitä, miksi vähäkoodinen ohjelmistokehitys on kasvunsa ja hypetyksensa ansainnut. He nostivat esiin, että yksi vähäkoodisen ohjelmistokehityksen isoista hyödyistä on se, että ratkaisuvaihtoehtoja pystytään toteuttamaan nopeasti ja näin näyttämään asiakkaille ja muille sidosryhmille, kuinka sovellus voisi toimia. Nopeuden ja ketteryyden ansiosta vähäkoodista ohjelmistokehitystä pystytään hyödyntämään myös sellaisiin hankkeisiin, jotka jäisivät muuten tekemättä, mikäli ne pitäisi tehdä perinteisesti koodaamalla.

Näen perinteistä ohjelmistokehitystä nopeampana ja esimerkiksi asiakkaan tai projektipäällikön kanssa mä oon kokenut että... pystyy hyvin näyttämään nopeastikin ratkaisuja tai ratkaisuvaihtoehtoja... vaikkei low-code koodia näyttäis, niin ylipäätänsä koska pystyy tekemään esim UI:ta ja näin poispäin, niin pystyy nopeastikin näyttämään erilaisia ratkaisuja ja tekemään esim POCia lennosta. - Low-code kehittäjä, 44

Low-code välineillä on hyvä toteuttaa kaikkia semmosia kehityshankkeita, mihin ei välttämättä sen suurempaa sovelluskehitystä haluttais muuten toteuttaa. Kun kaikki sovelluskehityshankkeet on suhteellisen isoja ja työläitä, niin jos sovelluksella saavutettavat hyödyt on suhteellisen pienehköt tai nettohyötyjänä on vaan muutama henkilö, niin isompi ohjelmistokehityshän ei sillon oo kauheen kannattavaa. Mut taas low-code työkalulla pystyttäis tällaisissa tilanteissa toteuttamaan jotain työtä tehostavan välineen. - Low-code kehittäjä, 27

Helpommin lähestyttävä ohjelmistokehityksen väline

Vähäkoodinen ohjelmistokehitys koettiin olevan helposti lähestyttävä ja ymmärrettävä ohjelmistokehityksen väline. Haastatteluissa nousi esiin se, että vähäkoodisen ohjelmistokehityksen avulla myös perinteiseen ohjelmistokehitykseen eli koodaamiseen voi päästä helpommin kiinni.

Näen [low-coden] tosi hyvänä työvälineenä nimenomaan etenkin siihen, kun low-code on semmonen helpommin lähestyttävä devausväline ja just semmonen helpommin ymmärrettävä. Näen [low-codella] todellakin potentiaalin tuoda devaamisen lähemmäs ihmistä. - Low-code kehittäjä, 27

Uskon että low-code voi olla hyvä myös siihen, että jos henkilö on kiinnostunut ohjelmistokehityksestä, niin tutustumalla low-codeen ohjelmistokehitys ei välttämättä tunnu niin kaukaiselta ja monimutkaiselta, vaan siitä voi saada helpommin kiinni myös ihan sen takia, että sitä omaa kädenjälkeä näkee heti. - Ohjelmistokehittäjä 24

4.2.2 Tekoäly

Lähes kukaan haastateltavista ei mieltänyt tekoälyä helposti lähestyttäväksi. Tekoälyä pidettiin haastattelijoiden keskuudessa erittäin korkealentoisena ja asiana, jota on vaikea käsittää.

En sanoisi tekoälyä lähestyttäväksi. Jos tulisi projektissa tehtävä, että pitäisi toteuttaa tekoälyä, tulisi alussa varmasti fiilis että osaako sitä ja ymmärtääkö tarpeeksi. Varmasti siitä sitten suoriutuisi mutta eka fiilis olisi tosi poispäin työntävä. - Low-code kehittäjä, 44

Olen ehdottomasti varautunut enkä koe olevani mukavuusalueellani tästä puhuttaessa. Se ehkä kertookin, että en koe helposti lähestyttäväksi. Tekoälyn eettisyys myös epäilyttää, etenkin jos käyttää paljon historiallista dataa. - HR, 46

Niin ja näin, vaikea kyllä sanoa [kokeeko tekoälyn lähestyttäväksi]. Kiinnostaa kyllä se mitä se mahdollistaa, mutta ei ole mitään intohimoa kurkistaa pinnan alle mitä siellä tapahtuu. - HR, 33

Epämääräisyys

Tekoälyä pidettiin myös etäisenä ja epämääräisenä käsitteenä. Haastateltavan kohderyhmä (ohjelmistokehittäjä vai joku muu) ei vaikuttanut siihen käsitykseen, että haastateltavalle ei ollut selvää, missä menee tekoälyn raja ja mitkä asiat luetaan tekoälyksi.

Joo-o, mikä sitten on tekoälyä? Jonkinlainen algorytmin tekemä päätös. Tuntuu kyllä jotenkin tosi isolta asialta joka tuntuu sellaiselta jota ei oikeen ymmärrä. - Low-code kehittäjä, 44

Tekoäly herättää omalla tavallaan ristiriitaisia fiiliksiä, koska määritelmä on niin laaja ja ei oo selvä mitä on tekoäly ja mitä ei. Itsekin opiskellut ja käynyt koulukursseja ja oon itse vielä kaiken lisäksi kiinnostunut osa-alueesta ja toteuttanut tekoälyä, mutta silti samalla tuntuu ettei kuitenkaan ymmärrä eikä osaa sanoa mikä nykyään luokitellaan tekoälyksi. - Ohjelmistokehittäjä 24

Hyötyjä vaikea tunnistaa

Tekoälyn käyttö ja hyödyllisyys herättivät ajatuksia haastateltavissa. Tekoälyn hyödyt ovat jääneet monilta tunnistamatta. Tekoälyn koettiin olevan käytössä joissain järjestelmissä vain teknologian takia.

Teknologiaa ei mun mielestä kannata tehdä vain teknologian takia. Omasta mielestä jos mennään teknologia edellä ja tehdään tekoälyä vaan sen takia, että tekoälyä saadaan mukaan, niin se ei oo hyvä. Tulisi miettiä miksi niitä teknologioita oikeesti hyödynnetään ja tuoko ne jotain hyötyä. - HR, 54

Siihen liittyy ehdottomasti hypeä ja kaikkeen halutaan lisätä sitä, niinkuin se olisi ratkaisu kaikkeen. Kunhan saadaan laitettua, että sovelluksessa hyödynnetään tekoälyä, niin sitten tuntuu olevan hyvä. Sama asia vaikka ML tai deep learningin kanssa. - Ohjelmistokehittäjä 24

Hyvä periaate

Tekoälystä löydettiin myös hyviä puolia. Haastateltavat kokivat, että tekoälyllä on monia käyttötarkoituksia ja sillä voidaan parantaa tehokkuutta monissa asioissa.

Periaate on hyvä, että hyödynnetään siihen että saadaan parempia ratkaisuja. On varmasti monia asioita, joita on parannettu hyödyntämällä tekoälyn tarjoamaa automatisaatiota. - Low-code kehittäjä, 44

On hienoa että tekoälyn avulla monia manuaalisia tehtäviä saadaan minimoitua ja työntekijät voivat keskittyä olennaiseen. Silloin kun sitä käytetään tällaisiin asioihin. Ja tehtäviä, joita tekoäly suorittaa, ovat hyvin yksinkertaisia eikä vaadi mitään ihmisajattelua vastaavaa. Eli pelkästään dataan perustuvaa, niin silloin mun mielestä kyllä hyvä asia - HR, 33

4.2.3 Visuaalisuuden vaikutus tekoälyn ymmärrettävyyteen ja selitettävyyteen

Haastatteluissa selvitettiin graafisten komponenttien vaikutusta tekoälyn ymmärrettävyyteen ja selitettävyyteen. Kaikki haastateltavat totesivat graafisten elementtien auttavan heitä hahmottamaan ja ymmärtämään ratkaisua paremmin. Esimerkiksi prosessien hahmottamisen koettiin olevan helpompaa, kun prosessit kuvataan graafisilla elementeillä. Haastateltavat ymmärsivät, miten tutkimuksessa toteutettu sovellus toimii.

Vaikka en itse työskentele devaajana, niin ymmärrän silti prosesseja ja järjestelmän arkkitehtuuria, mikäli sen joku esittää graafisesti... myös jos en esimerkiksi ymmärrä jotain kokonaisuutta, saatan hyvin piirtää sen auki itselleni. - Projektipäällikkö, 33

Mikäli mahdollista, niin käytän kyllä visuja ymmärtämiseen. Itselle visuaalinen tulkinta on tosi avaava. Esimerkiksi kun alkoi käyttämään symboleja ja kuvaajia matikassa, niin aloin heti ymmärtämään niitä paremmin. Myös koska oon tekoälyynkin perehtynyt, niin kyllä mulle avaa paljon se jos jotain tekoäly tai ML-prosesseja on kirjoitettu eri symbolien avulla eikä niin, että on vaan pitkiä matemaattisia pötköjä. Tällöin ymmärrän paremmin, että mihin se lopputulos sitten perustuu kun sen näkee selkeästi. - Ohjelmistokehittäjä 24

Koen olevani visuaalinen ihminen, ja ikonit ja kuvaajat tuntuu itselle sellaiselta, että niitä mieluusti käyttää. - HR, 33

Vähäkoodisen ohjelmistokehityksen graafiset mallit

Haastateltaville esiteltiin tutkimuksessa tehtyä vähäkoodista ohjelmisto toteutusta. Vähäkoodinen ohjelmistokehitys koettiin helposti lähestyttäväksi ja ymmärrettäväksi visuaalisuuden ja graafisten komponenttien ansiosta. Moni koki, että vähäkoodisen ohjelmistokehityksen logiikasta sai nopeasti kiinni jo demon aikana. Myös henkilöt, jotka eivät aikaisemmin olleet nähneet vähäkoodisen ohjelmistokehityksen graafisia malleja, ymmärsivät nopeasti, miten järjestelmä rakentuu.

Prosessikaavio tuntui tosi selkeältä ja sain heti if-else logiikasta ymmärrettyä, miten ne toimii ja mitä siinä tapahtuu. Neliöt oli if-lausekkeita ja elset oli niitä ympyröitä. Voisin hyvin piirtää vastaavaa jos haluaisin selventää itselleni jotain prosessia, mitä omassa projektissa tehdään. - Projektipäällikkö, 33

Joo [low-code] on kyllä helpompi lukuisempaa kuin [kirjoitettu] koodi. Menisi varmasti pidempään saada kiinni kirjoitetusta koodista, vaikka itselläkin koodaajataustaa on reilu viisi vuotta. - HR, 33

Polkujen hahmottamiseen koen, että [low-code] on tosi hyvä ja olit mun mielestä noi prosessit rakentanut tosi selkeästi ja ymmärrän kyllä hyvin mitä siinä tapahtuu... Toki riippuu myös ihmisestä, joka low-codeakin tekee, joillakin voi olla tosi sotkuisa tyyli miten rakentaa prosesseja jolloin kestää tietenki pidempää ymmärtää miten kaikki yhteydet menee ja mikä elementti tulee minkin jälkeen. Noin ylhäältä alas rakennettuna graafiset elementit pääsi oikeuksiinsa. - Low-code kehittäjä, 44

Tekoälyn selitettävyys

Haastattelussa perehdyttiin myös tekoälyn selitettävyyteen. Haastateltavilta kysyttiin, ymmärsivätkö he mihin tekoälyn tekemät päätökset perustuvat. Demon jälkeen haastateltavia pyydettiin vastaamaan, mihin heidän mielestään tekoäly perusti päätöksensä. Kaikki haastateltavat ymmärsivät, että työpaikkojen suosittelu -sovelluksessa käytetty tekoäly antaa suosituksensa perustuen pelaajan itse antamiin tietoihin (työkokemus ja koulutustausta) sekä pelistä saatuihin pisteisiin. Muutamat haastateltavat osasivat osoittaa oikeat kohdat, kun kysyttiin missä prosessin kohdassa tekoälyä käytetään.

Ymmärsin joo... [selitystä mihin tekijöihin tekoälyn logiikka perustui]. Mulla on kokemusta samankaltaisista prosesseista mutta silti uskon, että olisin voinut ymmärtää hyvin vaikkei tämmöistä kokemusta olisi. Että vaikka tosiaan kokemusta Power Platformista eikä OutSystemsistä, niin pystyi välittömästi tajuta missä mennään, mitä tapahtuu ja mitä tekijöitä tässä käytettiin. - Low-code kehittäjä, 27

Kyllä ymmärsin täysin. Täysin looginen mulle. If-logiikka oli tosi selkeä ja ymmärrettävä tolleen esitettynä. - HR, 54

Selitettävyyteen vaikuttavat tekijät

Haastateltavilta kysyttiin, mitkä tekijät vaikuttavat siihen, että he ymmärsivät tekoälyn tekemän päätöksen. Ymmärrykseen vaikutti selkeästi eniten prosessissa käytetty data. Haastateltavat kokivat, että kun heillä on ymmärrys sovelluksessa käytetystä datasta ja sen alkuperästä, heidän oli helpompi ymmärtää kokonaisprosessia.

Alkuspeksit vaikuttavat tosi paljon mulla, se että tietää mitkä on ne tekijät joita käytetään. Sitku on siitä pohjaa, nii sitten pystyy avaamaan että miten nää yksittäiset tekijät vaikuttamaan. - Ohjelmistokehittäjä 24

Data mihin päätökset perustuu, on isoin tekijä minkä ymmärryksen jälkeen koen että ymmärrän myös päätöstä paljon paremmin. Itelle on tärkeää ymmärtää myös miten se data on kerätty, että onko käyttäjän antamaa inputia, valmiista tietokannoista ja niin edespäin. - Projektipäällikkö, 33

Lähtöasetelma kokonaisuudessaan, että mitkä tekijät vaikuttavat ja miten niitä tekijöitä käsitellään, se lisää ehdottomasti ymmärrystä. - Low-code kehittäjä, 44

Ymmärrettävyyteen vaikuttavat tekijät

Haastateltavat nostivat esiin myös sen, että heidän ymmärrykseen vaikutti se, miten sovelluksen esittely tehtiin. Ymmärrys ei siis synny pelkästään visuaalisuuden ja graafisten komponenttien avulla, vaan ymmärrykseen vaikutti myös demon pitäjän vähäkoodisen ohjelmistokehityksen osaaminen.

Koen, että ymmärsin tosi hyvin [toteutuksen logiikkaa]. Vaikutti myös paljon se, että selitit järjestelmän logiikkaa hyvin. Näin sain tosi nopeasti kiinni. Toki olisi mennyt hieman pidempää, mikäli mun olisi itse pitänyt tutustua logiikkaan ilman selitystä. - Low-code kehittäjä, 44

Hyvän selityksen kanssa noi graafiset kaaviot ovat esimerkiksi monille asiakkaille varmasti paljon ymmärrettävämpiä. - HR, 33

Luottamus vähäkoodisella ohjelmistokehityksellä tuotettuun tekoälyyn

Haastateltavilta kysyttiin myös sitä, voivatko he luottaa tekoälyn tekemään päätökseen, vaikka he eivät täysin ymmärtäisi vähäkoodisen ohjelmistokehityksen komponenttien takana olevaa logiikkaa. Haastateltavat olivat yksimielisiä siitä, ettei tämä vähennä heidän luottamustaan tekoälyä kohtaan. Vaikka komponenteista ei suoraan näy mitä niiden takana tapahtuu, voi tekoälyn toimintaan päästä käsiksi vähäkoodisten kehitysalustojen tarjoamien lokien avulla. Henkilöt, joita asia kiinnostaa, voivat käydä sen selvittämässä.

Se, ettei näe sinne taakse, niin en näe että se olisi huono asia. Ei kaikkien tarvitse nähdä mitä siellä tapahtuu ja näen sen niin että ymmärrettävyys ei välttämättä ainakaan lisäänny jos näytetään taustatietoja. Joillekin se voi sekottaa pakkaa entisestään. En näe, että kaikkien tarvitsisi ymmärtää syvemmin... Itse voin ainakin luottaa komponentteihin, toki oman luonteen takia tykkään tutkia syvemmin asioista ja siihen OutSystems tarjoaa tosiaan Lifetimessa mahdollisuuden. - Low-code kehittäjä, 44

Ei yhtään haittaa [vaikka ei näe komponenttien taakse]. Päinvastoin, oon ilonen etten näe mitä kaikkea siellä on. - HR, 54

Riippuu kontekstista, mutta ei kyllä lähtökohtaisesti haittaa. Ainakin ton sun esityksen pohjalta ei haittaa ollenkaan, vaikka en tiedä asioita siellä taustalla. - Projektipäällikkö, 33

4.3 Yhteenveto empiirisen tutkimuksen tuloksista

Tässä kappaleessa vedetään yhteen empiirisen tutkimuksen tärkeimmät tulokset. Empiirinen tutkimus toteutettiin tapaustutkimuksen sekä puolistrukturoitujen haastattelujen avulla. Tapaustutkimuksessa käytettiin olemassa olevaa rekrytointipeliä, johon toteutettiin OutSystems-kehitysalustalla tekoälyä hyödyntävä työpaikkojen suosittelu -osio. Pelissä käyttäjä ratkaisee koodaustehtäviä neljällä eri ohjelmointikielellä. Tehtävien jälkeen pelaaja voi vastata työkokemukseen ja opiskelutaustaan liittyviin kysymyksiin, ja yhdessä taustatietojen ja pelistä saatujen pisteiden perusteella pelaaja saa suosituksia siitä, mihin Solitan avoimeen työtehtävään hänen tiedoillaan ja osaamisellaan kannattaisi hakea. Tapaustutkimuksella pyrittiin vastaamaan tutkimuskysymykseen "miten vähäkoodisella ohjelmistokehityksellä luodaan tekoälyä hyödyntäviä ohjelmistoja". Kyseiseen tutkimuskysymykseen saatiin vastauksia myös haastattelujen avulla.

Sovelluksen kehittäminen aloitettiin kuvaamalla yhteistyössä tilaajan kanssa sovelluksen toiminnot. Tilaajan vaatimukset kirjattiin ylös ja niiden pohjalta määriteltiin käyttäjäroolit, käyttötapaukset ja sovelluksen käyttämät tiedot. Tämän jälkeen aloitettiin kehitystyö OutSystems-kehitysalustalla. Ensimmäisenä kehitettiin työkokemuksen, koulutustaustan ja avoimen työpaikan tietomallit sekä tietokantarakenteet. Sitten toteutettiin tekoälyä hyödyntävä suosittelu-osio, jossa hyödynnettiin OutSystemsin sisäänrakennettuja toiminnallisuuksia. Suosittelu-osioon lisättiin käsinkoodaamalla koodia, jotta osio saatiin toimimaan määrittelyn mukaisesti. Käyttöliittymiä ei piirretty etukäteen, vaan ne toteutettiin suoraan OutSystemsiin. OutSystemsissä toteutusta oli helppo muuttaa ja eri vaihtoehtoja pystyi kokeilemaan ketterästi. OutSystems avusti monessa toteutustyön vaiheessa esimerkiksi luomalla tietokannasta valmiit UI komponentit.

Puolistrukturoidun haastattelun ensisijaisena tavoitteena oli selvittää, miten vähäkoodisessa ohjelmistokehityksessä käytetyt menetelmät vaikuttavat tekoälyn selitettävyyteen ja ymmärrettävyyteen. Haastatteluihin osallistui yhteensä seitsemän henkilöä ja he edustivat Solitan HR:ää, vähäkoodisen ohjelmistojen kehittäjiä, perinteistä ohjelmointia ja projektinhallintaa. Haastatteluilla pyrittiin vastaamaan tutkimuskysymykseen "miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta". Haastatteluissa esitettiin aiheeseen liittyviä kysymyksiä ja demonstroitiin rekrytointipeliä ja siihen toteutettua lisäosaa. Haastateltaville esiteltiin lisäosan taustalla toimivat vähäkoodisen ohjelmistokehityksen prosessit, jotta voitiin selvittää miten visuaalisuus vaikuttaa tekoälyn ymmärrettävyyteen ja selitettävyyteen.

Haastattelut vahvistivat ja täydensivät tapaustutkimuksen tuloksia. Haastateltavat kokivat, että vähäkoodinen ohjelmistokehitys on helposti ymmärrettävää graafisten komponenttien takia. Henkilöt, joilla ei ollut aiempaa kokemusta vähäkoodisesta ohjelmistokehityksestä, ymmärsivät nopeasti miten järjestelmä rakentuu ja toimii. OutSystems on kätevä väline toteuttaa tekoälyä hyödyntäviä ohjelmistoja, eikä välineen opetteluun mennyt pitkään. Välineen haltuunoton nopeuteen vaikutti se, että minulla on taustaa ohjelmistokehityksestä niin opintojen kuin työkokemuksen kautta. Ohjelmistokehityksen prosessi ja tehtävät olivat jo valmiiksi tuttuja. Välineen opetteluun ja tekoälyn toteutukseen meni yhteensä noin viisi työpäivää. Toteutus eteni nopeasti eikä muutoksia virheiden takia juurikaan tarvinut tehdä. Aluksi tein tekoälyyn vaadittavia tietojen siirtoja liian monimutkaisesti, kunnes opin miten kyseiset komponentit saa toteutettua OutSystemsillä hyvinkin yksinkertaisesti. Kaiken kaikkiaan koen, että OutSystemsillä tekoälyn toteuttaminen oli oikeasti nopeaa ja selkeää. Toki tulee huomioida, että tutkimuksessa toteutettu tekoäly oli hyvin yksinkertainen. Mikäli tekoälystä olisi haluttu tehdä monimutkaisempaa, olisi aikaa mennyt enemmän. Jos sovelluksen tekoälyä halutaan jatkokehittää, uskon ettei minun ei tarvitse opiskella Outsystemsin puolelta juurikaan uutta. Jatkokehitys voitaisiin tehdä jo valmiiksi opittujen komponenttien avulla.

OutSystems on vähäkoodinen kehitysalusta, jossa on mukana tekoälyavustaja. Vähäkoodisen ohjelmistokehityksen hyötyinä pidetään kehittämisen nopeutta ja helppoutta. Tapaustutkimuksessa todistettiin, että ohjelmiston kehittäminen OutSystemsin avulla on nopeaa ja ketterää. Nopeus perustuu graafisiin komponentteihin eli siihen, ettei kehittäjän tarvitse kirjoittaa koodia juuri lainkaan. Tutkimuksessa koodausta tarvittiin vain komponenttiin, joka tarkistaa onko pelaajalla työpaikkaan vaadittavaa työkokemusta sekä tarkastukseen, vastaako koulutustausta työpaikan vaatimaa koulutusvaatimusta. Molemmissa tapauksissa vaadittava koodi oli yhden rivin pituinen ja sisälsi vain kolme kohtaa.

EU:n asetusehdotuksessa tekoälyjärjestelmät luokitellaan neljään riskiryhmään käyttötarkoituksen perusteella. Työpaikkojen suosittelu -sovellus sijoittuu vähäisen riskiluokan kategoriaan ja siinä riittää, että käyttäjä tietää tekoälyn olemassaolosta. Sovellusta käytettäessä pelaaja ymmärtää, ettei hän on tekemisissä koneen kanssa, ei ihmisen.

5 Pohdinta

Tässä luvussa esitetään yhteenveto kuhunkin tutkimuskysymykseen liittyvistä tärkeimmistä tuloksista ja arvioidaan kriittisesti, miten hyvin tutkimuskysymyksiin pystyttiin vastaamaan. Luvussa kiteytetään empiirisen tutkimuksen tulokset ja verrataan niitä kirjallisuuskatsauksen tuloksiin.

5.1 RQ1: Vähäkoodisella ohjelmistokehityksellä tuotettu sovellus

Tässä kappaleessa vedetään yhteen diplomityön tulokset, jotka liittyvät tutkimuskysymykseen "miten vähäkoodisella ohjelmistokehityksellä luodaan tekoälyä hyödyntäviä sovelluksia". Vähäkoodinen ohjelmistokehitys (low-code) tarkoittaa sitä, että ohjelmistoja kehitetään graafisten mallien avulla siten, että perinteistä koodia tarvitaan joko vähän tai ei lainkaan [40]. Vähäkoodinen ohjelmistokehitys tapahtuu vähäkoodisilla kehitysalustoilla (low-code development platforms) [2]. Vähäkoodiset kehitysalustat tarjoavat ympäristön sovelluksen koko elinkaarelle mukaan lukien kehitys, toteutus, käyttöönotto ja ylläpito [43]. Tutkimuksessa käytettiin OutSystems Platform -kehitysalustaa, joka sisältää kaikki yllä mainitut elinkaaritoiminnot.

Tutkimuksessa toteutettiin OutSystems-kehitysalustalla yksinkertainen tekoälyä hyödyntävä työpaikkojen suosittelu -sovellus. Kehitys, toteutus ja testaus oli helppo tehdä OutSystemsiin sisäänrakennettujen elementtien avulla ja toteutus saatiin tehtyä erittäin nopeassa aikataulussa. OutSystemsin elementtejä ovat esimerkiksi if, for each, switch, assign ja run server action. Tämän lisäksi tekoälyn toteuttamiseen tarvittiin tiedonsiirtoja sekä tietokannoista että käyttäjän antamista tiedoista. Nämä tiedonsiirrot toteutettiin OutSystemsin valmiiden komponenttien avulla. Vaikka valmiit elementit tuodaan graafisilla Vedä ja pudota -komponenteilla, ehdot koodattiin manuaalisesti perinteisen koodin avulla. Yksinkertaisten ehtojen luomiseksi OutSystems tosin ehdottaa mahdollisia vaihtoehtoja. Tässä tutkimuksessa toteutetussa sovelluksessa kirjoitetun koodin osuus oli pieni. Kirjoitettua koodia tuli koko sovellukseen ainoastaan kolme riviä.

Vähäkoodisen ohjelmistokehityksen hyötyinä pidetään kehittämisen nopeutta ja helppoutta [2], ja tähän lopputulokseen tultiin myös tapaustutkimuksessa. Organisaatiot voivat julkaista uusia tai päivitettyjä tuotteita nopeasti eikä kehitykseen kulu aikaa kuukausia tai jopa vuosia. Tosiasia kuitenkin on, ettei kehitysväline itsessään tee kehittämisestä helppoa, vaan vaikeusaste tulee ratkaistavan ongelman monimutkaisuudesta. Myös vähäkoodiseen kehittämiseen kannattaa ottaa IT-asiantuntijoita mukaan. Varsinkin jos

ohjelmisto sisältää rajapintoja muihin palveluihin, sovellusarkkitehtuuria osaavia henkilöitä on syytä ottaa mukaan. Tutkimuksen haastatteluissa nousi esiin IT-asiantuntijan (tässä tapauksessa low-code kehittäjän) tärkeä rooli. Vähäkoodisella kehitysalustalla toteutetun sovelluksen toiminnan ymmärrys ei synny pelkästään visuaalisuuden ja graafisten komponenttien avulla, vaan ymmärrykseen vaikutti myös sovellusta demonstroineen kehittäjän OutSystems-osaaminen.

Vähäkoodinen ohjelmistokehitys ei poista sovelluskehitysprojektin tarvetta, vaan myös siinä tarvitaan vaatimusten määrittelyä, sovelluksen suunnittelua ja testausta. Selkeät roolit ja päätöksentekoprosessi korostuvat varsinkin silloin, kun kehittäminen tapahtuu nopeasti. Tutkimuksessa sovelluksen kehittämisen vaativin osa oli tietomallin hahmottaminen. Vähäkoodisen kehitysalustan hyvänä puolena on se, ettei kerralla tarvitse olla valmista, vaan sovelluksen toteutuksessa voitiin edetä ketterästi iteroiden. Väli- ja lopputuloksia pystyi arvioimaan ja testaamaan kehityksen kaikissa vaiheissa. Toinen hyvä puoli vähäkoodisessa kehittämisessä oli se, että OutSystems auttoi kehittämistyössä ja ehdotti oikeita työkaluja ja malleja sekä pyrki näin ehkäisemään virheiden syntyä.

Ensimmäiseen tutkimuskysymykseen onnistuttiin vastaamaan hyvin. Vaikka toteutettu sovellus oli toiminnoiltaan yksinkertainen ja tutkimuksessa käytettiin ainoastaan yhtä vähäkoodista kehitysalustaa, voitiin tutkimuksella todistaa, että vähäkoodisella kehitysalustalla voidaan luoda tekoälyä hyödyntävä sovellus. Myös monimutkaisemmat ohjelmistot käyttäisivät samoja komponentteja ja malleja kuin tässä sovelluksessa käytettiin - tosin laajemmassa mittakaavassa. Tutkimuksessa pystyttiin hyödyntämään OutSystemsin sisäänrakennettuja toiminnallisuuksia ja automaatioita.

5.2 RQ2: Vähäkoodisen kehitysalustan graafisten mallien vaikutus tekoälyn selitettävyyteen ja ymmärrettävyyteen

Toinen tutkimuskysymys oli, miten vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit vaikuttavat tekoälyn ymmärrettävyyteen ja selitettävyyteen ohjelmiston tilaajan kannalta.

Tekoälyn ymmärrettävyydellä tarkoitetaan sitä, että ihminen pystyy ymmärtämään ohjelmiston toimintaa [15]. Ohjelmiston toiminnalla tarkoitetaan tässä ohjelmiston toiminnallisuuksia, siinä käytettävää dataa ja sitä, millaisia integraatioita ohjelmistolla on muihin järjestelmiin. Selitettävyydellä tarkoitetaan, että ihminen ymmärtää mihin tekoälyn tekemät päätökset

perustuvat. Ihmiselle on tärkeää ymmärtää, miten tekoälyn päätöksentekomalli on luotu sekä miten ja kuka sitä on kouluttanut [6]. Black-box malleissa tekoälyn päätöksentekomalli ei avaudu eikä ihminen tunnista mihin päätökset perustuvat [30]. Luottamus tekoälyn päätöksentekomalliin nähdään erityisen tärkeänä silloin, jos päätökset vaativat tulkintaa tai jos ympäristö on epävarma tai monimutkainen.

Selitettävyys ei ole tärkeää vain päätösten perustelemiseksi, vaan se voi myös ehkäistä virheitä ja auttaa virheiden korjaamisessa. Kun käyttäjä ymmärtää tekoälyjärjestelmän toimintaa, hän voi tunnistaa siinä olevia virheitä ja puutteita.

Vähäkoodinen ohjelmistokehitys perustuu valmiisiin graafisiin Vedä ja pudota -komponentteihin. Tutkimuksen haastatteluissa korostui graafisten mallien tärkeys vähäkoodisten tekoälyjärjestelmien toiminnan ymmärtämisessä. Toteutettua lisäosaa demonstroitiin haastateltaville henkilöille haastatteluiden yhteydessä. Samalla esiteltiin kyseisen lisäosan taustalla toimivat OutSystems-prosessimallit, jotta voitiin selvittää miten visuaalisuus vaikuttaa tekoälyn ymmärrettävyyteen ja selitettävyyteen. Haastateltavat olivat yksimielisiä siitä, että graafisia prosessikaavioita oli helppo seurata ja OutSystems-ohjelmistokehitystä oli helpompi ymmärtää kuin perinteistä koodipohjaista kehittämistä. Kaikki haastateltavat osasivat sovelluksen ja prosessikaavioiden esittämisen jälkeen vastata siihen, mihin tekoälyn tekemät päätökset perustuvat. Haastateltavat ymmärsivät OutSystemsillä tuotetun tekoälyn päätöksenteon logiikan.

Ymmärrys sovelluksen toiminnasta voi kuitenkin olla virheellistä. Mitä enemmän komponenteista tulee graafisia, sitä syvemmäksi black-box muuttuu. Tämä johtuu siitä, että käyttäjä ei välttämättä tiedä, millaista logiikkaa vähäkoodisen kehitysalustan komponenttien takana on. Empiirisessä tutkimuksessa pyrittiin selvittämään, luottavatko ihmiset vähäkoodisella kehitysalustalla toteutettuun tekoälyyn, vaikka he eivät tietäisikään mitä toiminnallisuuksia OutSystemsin sisäisten komponenttien takana on. Ainakin haastattelujen perusteella voidaan päätellä, ettei tämä häiritse käyttäjiä. Vähäkoodisen kehitysalustan avulla on mahdollista tarkastella komponenttien toimintaa kooditasolla, mikäli henkilö - esimerkiksi kehittäjä - näin haluaa. Kehittäjälle tarjotaan siis mahdollisuus saada tietoonsa se, millaista logiikkaa komponenttien takana on. Tällä voidaan ehkäistä black-boxin syntymistä. Toinen keino black-boxin ehkäisemiseksi on sovelluksen testaus. Vaikka kehittäjä ei täysin ymmärtäisi komponenttien sisäänrakennettua toimintaa, hän voi helposti testata niitä ja ymmärtää näin paremmin tekoälyn logiikkaa.

Tekoälyjärjestelmän käyttäjän on myös ymmärrettävä, mitä tekoäly tekee ja mitä käyttäjältä itseltään vaaditaan [6]. Rekrytointipelin pelaajan tulee ymmärtää, että sovelluksen tekemä suositus perustuu osin hänen antamiinsa taustatietoihin. Jos nämä taustatiedot (koulutustausta, työkokemus) ovat virheellisiä tai puutteellisia, tekoälyn suositukseen ei voi luottaa. Sovelluksessa käytetyn datan tärkeys korostui myös haastatteluissa. Haastateltavien mielestä tekoälyn selitettävyys paranee, kun käyttäjä pystyy tunnistamaan ohjelmistossa käytettävän datan ja ymmärtää, mistä data on peräisin.

Haastatteluissa käytiin sovelluksen tilaajan eli Solitan HR:n kanssa läpi rekrytointipelin ja työpaikkojen suosittelu -osion toteutusta. Tilaaja oli hyvin tyytyväinen toteutukseen ja ymmärsi, mihin tekoälyn tekemät päätökset perustuivat.

Toiseen tutkimuskysymykseen onnistuttiin vastaamaan kohtuullisesti. Vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit edistävät tekoälyn ymmärrettävyyttä ja selitettävyyttä. Käyttäjien on helpompi lähestyä vähäkoodisella kehitysalustalla tuotettua tekoälyä, sillä graafisten mallien avulla käyttäjät saavat logiikasta nopeammin kiinni ja useille visuaalinen kuva lisää ymmärrystä.

5.3 Tutkimuksen rajoitteet

Tutkimuksen rajoitteet koskivat suurimmalta osalta diplomityön tekemiseen rajattua aikaa, joka oli sovittu yhdessä kohdeyrityksen (Solita) kanssa. Yhtenä vaihtoehtona oli toteuttaa tutkimus asiakasprojektina, mutta sopivaa asiakasta ei löytynyt tutkimukseen varattuna ajanjaksona. Tästä syystä päädyttiin siihen, että tutkimuksessa käytetään Solitan olemassa olevaa rekrytointipeliä, johon toteutetaan OutSystems-kehitysalustalla tekoälyä hyödyntävä työpaikkojen suosittelu -osio. Mikäli käytettävissä olevaa aikaa olisi ollut enemmän, OutSystemsillä toteutetusta sovelluksesta olisi voitu tehdä monimutkaisempi ja se olisi voinut sisältää enemmän tekoälytekniikoita. Sovelluksessa olisi voinut hyödyntää laajemmin erilaisia OutSystemsin tarjoamia elementtejä ja työkaluja. Myös haastatteluja olisi voitu pitää laajemmalle joukolle, jotta olisi saatu laajempi otanta. Haastatteluja olisi voitu ajan salliessa laajentaa isommalle määrälle henkilöitä, eri rooleihin ja eri Solitan organisaatioihin.

Tämän diplomityön empiirinen tutkimus ei toteutetun sovelluksen pienuuden takia pureutunut syvälle tekoälyn ymmärrettävyyteen ja selitettävyyteen. Sovelluksessa käytettiin yksinkertaisia tekoälytekniikoita, jotka haastateltavien oli helppo ymmärtää ainakin selitettyinä. Esittelin haastateltaville tutkimuksessa toteutetun lisäosan taustalla toimivat OutSystems-prosessimallit.

On mahdollista, etteivät kaikki haastateltavat olisi ymmärtäneet niitä itsekseen. Toisaalta tämä todistaa sen, että myös vähäkoodiseen kehittämiseen kannattaa ottaa IT-asiantuntijoita mukaan.

Diplomityön tekemistä vaikeutti aikarajoitusten lisäksi se, että vähäkoodiseen ohjelmistokehitykseen liittyvä tieteellinen kirjallisuus painottuu vähäkoodisten kehitysalustojen teknisiin ominaisuuksiin. Ensimmäiseen tutkimuskysymykseen löytyi tieteellistä kirjallisuutta, mutta toiseen tutkimuskysymykseen ei niinkään. Vähäkoodisen ohjelmistokehityksen ymmärrettävyyttä tai selitettävyyttä ei ole juurikaan tutkittu, kuten ei myöskään kansalaiskehittämisen todellisia hyötyjä tai onnistumisia. Osa kirjallisuuskatsauksen aineistosta on peräisin blogeista, seminaareista ja IT-toimittajien materiaaleista. Näissä olevat tiedot eivät perustu tutkimusnäyttöön, vaan ovat kirjoittajan näkemyksiä tai arvioita asioista.

6 Johtopäätökset

Diplomityössä tunnistettiin tekoälyn ymmärrettävyyteen ja selitettävyyteen vaikuttavia tekijöitä olemassa olevan kirjallisuuden perusteella. Tämän jälkeen toteutettiin OutSystems-kehitysalustalla tekoälyä hyödyntävä sovellus, jonka toimivuutta arvioitiin kohdeyrityksen kontekstissa haastattelujen avulla. Tutkimuksen lopputuloksena tehtiin alla olevat johtopäätökset:

Vähäkoodiset ohjelmistokehitysalustat tarjoavat valmiita työkaluja ohjelmistojen suunnitteluun ja toteuttamiseen. Vähäkoodiset kehitysalustat hyödyntävät itsessään tekoälyä avustamalla kehittäjiä sovellusten kehitystyössä. Vähäkoodisilla kehitysalustoilla voidaan kuitenkin myös tuottaa tekoälyä sisältäviä sovelluksia kehitysalustaan sisäänrakennettujen elementtien avulla. Sisäänrakennettuja elementtejä ovat esimerkiksi erilaiset loopit, tietojen siirrot ja määrittelyjen elementit. Elementeistä rakennetaan prosesseja graafisten komponenttien avulla graafisen käyttöliittymän kautta.

Vähäkoodisen kehitysalustan avulla tuotetut graafiset mallit lisäävät tekoälyn ymmärrettävyyttä ja selitettävyyttä. Tutkimuksen mukaan vähäkoodisen kehitysalustan avulla toteutettu sovellus koettiin ymmärrettäväksi visuaalisuuden ja graafisten komponenttien ansiosta. Tekoälyn ymmärrettävyys paranee, jos ihminen pystyy ymmärtämään sovelluksen toimintaa ja siinä käytettävää dataa.

Tekoälyn selitettävyyteen vaikuttaa se, kuinka hyvin ihminen ymmärtää tekoälyn päätöksentekomallia. Tutkimuksessa luottamus vähäkoodisella kehitysalustalla tuotettua tekoälyä kohtaan ei vähentynyt, vaikka käyttäjä ei nähnyt komponenttien takana olevaa tekoälyn logiikkaa. Tätä black-box -ilmiötä voidaan ehkäistä muun muassa tekoälyssä käytettävän datan laadulla. Tekoälyn selitettävyys paranee, kun käyttäjä pystyy tunnistamaan ohjelmistossa käytettävän datan ja ymmärtää, mistä data on peräisin. Datalla on suuri merkitys myös silloin, kun tekoälyä koulutetaan.

Tutkimuksen edetessä tunnistin useita jatkotutkimuksen kohteita. Vähäkoodista ohjelmistokehitystä on tutkittu verrattain vähän aikaa, ja esimerkiksi vähäkoodisen ohjelmistokehityksen hyödyistä on niukasti tutkimusnäyttöä. Myös sen haittapuolia on syytä tutkia. Yksi kiinnostava tutkimusaihe voisi olla vähäkoodisella kehitysalustalla toteutetun sovelluksen integroiminen yrityksen muihin järjestelmiin; kuinka helposti vähäkoodisella kehitysalustalla voi toteuttaa rajapintoja ulkoisiin järjestelmiin ja kuinka paljon rajapintojen toteutus vaatii perinteistä ohjelmointia - vai vaatiiko lainkaan. OutSystemsin vuonna 2020 teettämän kyselytutkimuksen mukaan lähes viidennes suomalaisista uskoo koodaamisen tulevan osaksi tulevaisuuden työnkuvaa [45]. Tämän ennusteen toteutumista kannattaisi tutkia myöhemmin. Lisäksi

olisi mielenkiintoista tutkia, pystyvätkö loppukäyttäjät ja ei-ohjelmistokehityksen tai ei-teknisen koulutuksen saaneet henkilöt ymmärtämään OutSystems-alustan avulla luotuja prosessimalleja itsekseen.

Lähteet

[1] Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (XAI), IEEE, 52138–52160.

[2] Ahinaslan, E., Sahinaslan, O. & Sabancıoglu, M. (2021). Low-code application platform inmeeting increasing software demands quickly: SetXRM. AIP Conference Proceedings, AIP Publishing.

[3] Alonso, A. N., Abreu, J., Nunes, D., Vieira, A., Santos, L., Soares, T., & Pereira, J. (2020). Towards a Polyglot Data Access Layer fo Low-Code Application Development Platform.

[4] Alpaydin, E. (2016) Machine Learning: The New AI. Cambridge, MA: MIT Press.

[5] Arrow ECS Finland Oy (2021). Vähemmän koodia markkinakartoitus. Saatavissa: https://www.arrow.com/ecs-media/14448/2021-02-fi-infograph.pdf.

[6] Ashoori, M., Weisz, J. D. (2019). In AI We Trust? Factors That Influence Trustworthiness of AI-infused Decision-Making Processes.

[7] AVI Networks (2021). Platform as a Service. Saatavissa: https://avinetworks.com/glossary/platform-as-a-service.

[8] Brynjolfsson, E., McAfee, A. (2011). Winning the race with ever-smarter machines. MIT Sloan Management Review, 53(2), 53-60.

[9] Coeckelbergh, M. (2012). Can we trust robots? Ethics and Information Technology, 14(1), 53–60.

[10] Dane, E., Rockmann, K. W., & Pratt, M. G. (2012). When should I trust my gut? Linking domain expertise to intuitive decision-making effectiveness. Organizational Behavior and Human Decision Processes, 119, 187–194.

[11] Digital Illustrated (2021). 5 myyttiä low-code kehittämisestä. Saatavissa: https://digitalillustrated.com/5-myyttia-low-code-kehittamisesta.

[12] Du-Harpur, X., Watt, F. M., Luscombe, N. M., Lynch, M. D. (2020). What is AI? Applications of artificial intelligence to dermatology. British Journal of Dermatology, 183(3), 423-430.

[13] eCraft (2018). Mitä on low-code ja kuka sitä tarvitsee? Saatavissa: https://ecraftweb2016.square-pace.com/fin/blog/2018/2/15/mita-on-low-code-ketteraa-sovelluskehitysta.

[14] Euroopan komissio (2019). Luotettavaa tekoälyä koskevat eettiset ohjeet. 04/2019. Saatavissa: http://www.europarl.europa.eu/meetdocs/2014_2019/plmrep/COMMITTEES/JURI/DV/2019/11-06/Ethics-guidelines-AI_FI.pdf.

[15] Floridi, L., Cowls, J., Beltrametti, M., Chatila, R., Chazerand, P., Dignum, V., Vayena, E. (2018). AI4People—An Ethical Framework for a Good AI Society: Opportunities, Risks, Principles, and Recommendations. Minds and Machines, 28(4), 689–707.

[16] Forward Forever (2021). Low-code -alustojen mahdollisuudet PK-yrityksille. Webinaari 11.2.2021. Saatavissa: https://forwardforever.com/wp-content/uploads/2021/02/FF-Webinaari-2020-02-11-Low-code-alustojen-mahdollisuudet-PK-yrityksille.pdf.

[17] Gartner (2020). 2 Trends on the Gartner Hype Cycle for Artificial Intelligence. Saatavissa: https://www.gartner.com/smarterwithgartner/2-mega-trends-dominate-the-gartner-hype-cycle-for-artificial-intelligence-2020.

[18] Gartner (2021). Magic Quadrant for Enterprise Low-Code Application Platforms. Saatavissa: https://www.gartner.com/doc/reprints?id=1-275QSBDL&ct=210813&st=sb.

[19] Griffin, A. (2017). Facebook's AI creating its own language is more normal than people think, researchers say, The Independent.

[20] Gunning, D., & Aha, D. (2019). DARPA’s Explainable Artificial Intelligence (XAI) Program, AI Magazine, 40, 44–58.

[21] Hagras, H. (2018). Toward Human-Understandable, Explainable AI, Computer, 51(9), 28–36.

[22] Helsingin Sanomat (2021). Kooditon tulevaisuus. Niclas Storås 16.12.2021. Artikkeli.

[23] Holm, E. (2019). In defense of the black box. Science, 364(6435), 26–27.

[24] Holmström, J., Ketokivi, M. and Hameri, A. P. (2009). Bridging practice and theory: A design science approach. Decision Sciences, 40(1), 65-87.

[25] Jacinto, A., Lourenço, M., & Ferreira, C. (2020). Test Mocks for Low-Code Applications built with OutSystems. In ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (pp. 1-5).

[26] Jarrahi, M. H. (2018). Artificial Intelligence and the Future of Work: Human-AI Symbiosis in Organizational Decision Making, Business Horizons, 61 (4), 577-586

[27] Lyytinen, K. (2020). Yritysjohtaja – tiedätkö, miksi kansalaiskehittäminen on yksi 2020-luvun tärkeimmistä trendeistä? Talouselämä. 12.2.2020.

[28] Lyytinen, K. (2020). IT:ssä herätys - kansalaiskehittäjät tulevat! Efima. 29.1.2020.

[29] McCarthy, J. (2007). What is artificial intelligence? John McCarthy's home page. 12.11.2007.

[30] Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence 267, 1–38.

[31] Nordic Law (2021). Tekoäly saamassa uutta sääntelyä EU:ssa. 10.8.2021. Saatavissa: https://nordiclaw.fi/tekoaly-saamassa-uutta-saantelya-eussa.

[32] OutSystems, Outsystems platform features. Saatavissa: https://www.OutSystems.com/platform.

[33] OutSystems (2019). The State of Application Development. Is IT Ready for Distruption? OutSystems: Boston, MA, USA.

[34] Retool (2021). What is low code? A comprehensive guide. 25.6.2021. Saatavissa: https://retool.com/blog/what-is-low-code.

[35] Richardson, C., & Rymer, J. R. (2016). Vendor landscape: The fractured, fertile terrain of low-code application platforms. Forrester, Cambridge.

[36] Rossi, F., Ala-Pietilä, P., Bauer, W., Bergmann, U., Beliková, M., Bonefeld-Dahl, C., ... Yeung, K. (2019). A DEFINITION OF AI: MAIN CAPABILITIES AND DISCIPLINES. Bryssels: European Comission.

[37] Ruusuvuori, J., & Tiittula, L. (2005). Haastattelu. Tutkimus, tilanteet ja vuorovaikutus. Vastapaino.

[38] Sahay, A., Indamutsa, A., Di Ruscio, D., & Pierantonio, A. (2020). "Supporting the understanding and comparison of low-code development platforms," In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, 171–178.

[39] Samek, W., Wiegand, T., & Müller, K. (2018). Explainable Artificial Intelligence: Understanding, Visualizing and Interpreting Deep Learning Models. ICT Discoveries, 1 (1), 39–48

[40] Sanchis, R., García-Perales, Ó., Fraile, F., Poler, R. (2020). Low-Code as Enabler of Digital Transformation in Manufacturing Industry. Applied Sciences. 10(1), 12.

[41] Schneider, J., & Handali, J. (2019). Personalized explanation for machine learning: A conceptualization. 27th European Conference on Information Systems (ECIS 2019). Stockholm-Uppsala, Sweden.

[42] Solita (2021). Nopeampaa ohjelmistokehitystä automaation avulla, 30.3.2021. Saatavissa: https://www.solita.fi/blogit/nopeampaa-ohjelmistokehitysta-automaation-avulla.

[43] Tisi, M., Mottu, J. M., Kolovos, D., De Lara, J., Guerra, E., Di Ruscio, D., ... & Wimmer, M. (2019). Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms, 1st Junior Researcher Community Event, 67-76

[44] Työ- ja elinkeinoministeriö (2021). Valtioneuvoston kirjelmä eduskunnalle komission ehdotuksesta Euroopan parlamentin ja neuvoston asetukseksi tekoälyn harmonisoiduksi sääntelyksi (Artificial Intelligence Act) U 28/2021 vp. 27.5.2021. Saatavissa: https://www.eduskunta.fi/FI/vaski/Kirjelma/Sivut/U_28+2021.aspx.

[45] Uusi teknologia.fi (2020). Koodaamisesta osa työnkuvaa? 29.5.2020. Saatavissa: https://www.uusiteknologia.fi/2020/05/29/koodaamisesta-osa-tyonkuvaa.

[46] Wallenius, N. (2021). Kokonaisarkkitehtuuri – kaikki mitä aiheesta tarvitsee tietää. 23.2.2022. Saatavissa: https://niklaswallenius.fi/kokonaisarkkitehtuuri-taydellinen-opas.

[47] Wachter, S., Mittelstadt, B., & Russell, C. (2018). Counterfactual explanations without opening the black box: Automated decisions and the GDPR. Harvard Journal of Law & Technology, 31(2), 842- 861.

[48] Waszkowski, R. (2019). Low-code platform for automating business processes in manufacturing. IFAC-PapersOnLine, 52(10), 376–381.

[49] Woo, M. (2020). The Rise of No/Low Code Software Development-No Experience Needed? Engineering (Beijing, China) 6(9), 960-961.

[50] Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007), A design science research methodology for information systems research. MIS, 24 (3),45-77.

LIITTEET

Liite 1: Haastattelukysymykset

Diplomityö: haastattelurunko (45 min)

JOHDANTO (5-10 min.)

- Haastattelun periaatteet ja tavoitteet
  - Aineiston käyttö, GDPR (lomakkeen vastaukset, haastattelun muistiinpanot, tallenne ja käyttö)
- ikä, työpaikka, tekninen tausta, harrasteneisuus

YLEISTÄ LOW-CODE JA TEKOÄLY (10-15 min.)

- Millä tasolla low-code ohjelmistokehitys on sinulle tuttua?

- Mitä ajatuksia low-code herättää?

- Millaisena ohjelmistokehitys välineenä näet low-coden?

- Mitä ajatuksia sana tekoäly herättää?

- Koetko tekoälyn helposti lähestyttäväksi?

- Koetko graaffiset kuvat helpommin lähestyttävämmäksi kuin esimerkiksi tekstin?

REKRYTOINTI PELI JA LOW-CODE TOTEUTUS (20 min.)

- Rekrytointipelin ja sen kehityksen demoaminen

- Koetko, että ymmärrät esittämiäni graafisia kaavioita paremmin kuin kirjoitettua koodia?

- Ymmärrätkö, kuinka sovellus toimii ja missä sovelluksen osiossa on käytetty tekoälyä?

- Ymmärrätkö, miksi päätös tehtiin ja mitä tekijöitä käytettiin päätöstä tehtäessä? Mitkä tekijät mielestäsi vaikuttivat tekoälyn päätökseen?

- Mitkä tekijät vaikuttavat ymmärrykseesi siitä, mihin tekoälyn tekemä päätös perustuu?

- Mitä mieltä olet siitä, voitko luottaa siihen, että pelaajalle ehdotetaan parhaiten sopivaa työtehtävää annettujen tietojen pohjalta (eli voitko luottaa tekoälyn tekemään lopputulokseen)?

- Häiritseekö sinua, jos et täysin ymmärrä graafisen käyttöliittymän logiikkaa tai et ymmärrä mitä graafisen käyttöliittymän takana tapahtuu?

LOPUKSI (5 min)

- Yhteenveto tutkimuksesta ja nykyisistä löydöksistä

- Heräsikö yhteenvedosta vielä jotain aatteita low-coden ymmärrettävyydestä?

- Millä keinoilla (grafiikka, kuva, ohjeteksti tms) tekoälyn tekemää päätöstä voisi ymmärtää paremmin?

- Onko sinulla jotain kysymystä tai vastaustasi mielessä johon vaihtaisit vastaustasi yhteenvedon perusteella?