TURUN YLIOPISTO
Kauppakorkeakoulu

Low-code-alustojen käyttö tietojärjestelmäkehityksessä: mahdollisuudet ja uhat

Tietojärjestelmätieteen kandidaatintutkielma

Laatija:
Alpo Nykänen

Ohjaaja:
KTM Tarja Matikka

6.5.2026
Turku

Opiskelijan lausunto tekoälyn käytöstä tähän tutkielmaan liittyen:

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

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

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

Kandidaatintutkielma

Oppiaine: Tietojärjestelmätiede
Tekijä: Alpo Nykänen
Otsikko: Low-code-alustojen käyttö tietojärjestelmäkehityksessä: mahdollisuudet ja uhat
Ohjaaja: Tarja Matikka
Sivumäärä: 31 sivua + liitteet 1 sivu
Päivämäärä: 6.5.2026

Tiivistelmä

Tietojärjestelmäkehitys on kokonaisvaltainen, kaikki organisaation sidosryhmät osallistava jatkuva prosessi, jonka hoitaminen on kuitenkin viime vuosiin saakka edellyttänyt kattavaa tietoteknistä ymmärrystä ja ohjelmoinnin osaamista, minkä takia tekninen toteutus on ollut enimmäkseen ohjelmistokehityksen ammattilaisten vastuulla. Low-code-teknologia muuttaa tätä asetelmaa mahdollistamalla tietojärjestelmäkehitykseen osallistumisen ilman teknistä ohjelmistokehitysosaamista.

Tässä tutkielmassa pyritään kartoittamaan low-code-kehitykseen liittyviä mahdollisuuksia ja uhkia kolmen tutkimuskysymyksen avulla: 1) Miten low-code-alustat vaikuttavat tietojärjestelmäkehitysprosessin tehokkuuteen verrattuna perinteisiin tietojärjestelmäkehitysmenetelmiin? 2) Onko low-code-kehityksellä ja perinteisillä kehitysmenetelmillä tuotettujen järjestelmien laadussa eroja? 3) Mitä riskejä low-code-kehitykseen liittyy ja miten niitä voidaan hallita? Tutkielma toteutetaan kirjallisuuskatsauksena, jossa etsitään olennaisilla hakusanoilla aiheeseen liittyvää aineistoa tietojärjestelmätieteen ja tietotekniikan alan akateemisista perusjulkaisuista.

Kirjallisuuskatsauksen perusteella low-code-alustat tehostavat tietojärjestelmäkehitysprosessia muun muassa nopeuttamalla kehitystä ja parantamalla resurssitehokkuutta, mutta tuotetun järjestelmän laatu voi olla perinteisillä kehitysmenetelmillä tuotettua heikompi. Low-code-kehitykseen liittyy merkittäviä teknisiä, hallinnollisia, strategisia ja liiketoiminnallisia riskejä, jotka voivat pahimmillaan olla vahingoksi organisaatiolle. Johtopäätös on, että asianmukaisella implementoinnilla voidaan saavuttaa low-coden hyödyt sekä mahdollistetaan low-code-kehityksen ja perinteisen tietojärjestelmäkehityksen toteuttaminen rinnakkain.

Avainsanat: low-code, no-code, LCD, LCP, LCDP, tietojärjestelmäkehitys, kansalaisohjelmointi

SISÄLLYS

1 Johdanto

Tietojärjestelmäkehitys on kokonaisvaltainen, organisaation kaikki sidosryhmät osallistava jatkuva prosessi, jonka käytännön toteutus on kuitenkin viime vuosiin saakka vaatinut kattavaa tietotekniikan ja ohjelmoinnin osaamista, minkä takia se on ollut enimmäkseen ohjelmistokehityksen ammattilaisten vastuulla. Low-code-kehitys muuttaa tätä asetelmaa mahdollistamalla tietojärjestelmäkehitykseen osallistumisen ilman teknistä ohjelmistokehitysosaamista.

Low-code-kehitys on yleistyvä tietojärjestelmä- ja ohjelmistokehitysteknologia, joka pyrkii yksinkertaistamaan tietojärjestelmäkehitysprosessin helpoksi, visuaalisessa käyttöliittymässä toteutettavaksi tehtäväksi, johon aiemmasta poiketen pystyvät muutkin kuin teknisesti valveutuneet ohjelmistokehityksen ammattilaiset (Sahay ym. 2023). Näin ollen kaikki organisaation jäsenet saavat aiempaa kokonaisvaltaisemman tietojärjestelmäkehityskokemuksen, jossa he voivat vaikuttaa järjestelmän toiminnallisuuksiin ja käytettävyyteen konkreettisemmin kuin koskaan aiemmin. Vuoteen 2028 mennessä jopa 60 % ohjelmistokehitysyhtiöistä käyttää low-code-alustoja pääasiallisena sisäisenä kehitysalustanaan (Gao & Fagerholm 2026). Yksi suurimmista motivaattoreista low-code-kehityksen omaksumiseen on kehitysprosessin nopeuttaminen (Naqvi ym. 2025), mutta tätä teknologiaa otetaan käyttöön myös liiketoimintaprosessien parantamiseksi etenkin automaation näkökulmasta (Ajimati ym. 2025).

Low-code-kehityksellä on jo vahva jalansija tietojärjestelmäkehityksessä, mutta monien etujen ohella siihen liittyy merkittäviä haasteita, jotka liittyvät kehitysprosessiin, kehitettävän järjestelmän laatuun sekä organisaatioon itseensä. Mitkä ovat low-code-kehityksen mahdollisuudet ja uhat? Tähän kysymykseen pyritään vastaamaan tässä kirjallisuuskatsauksena toteutettavassa kandidaatintutkielmassa seuraavien tutkimuskysymysten avulla:

- Miten low-code-alustat vaikuttavat tietojärjestelmäkehitysprosessin tehokkuuteen verrattuna perinteisiin tietojärjestelmäkehitysmenetelmiin?
- Onko low-code-kehityksellä ja perinteisillä kehitysmenetelmillä tuotettujen järjestelmien laadussa eroja?
- Mitä riskejä low-code-kehitykseen liittyy ja miten niitä voidaan hallita?

Luvussa 2 määritellään tutkielmassa käytetyt keskeiset käsitteet, ja luvussa 3 esitellään teoreettinen viitekehys sekä siitä johdetut vertailukriteerit. Luvussa 4 tarkastellaan low-code-alustojen vaikutusta tietojärjestelmäkehitysprosessin tehokkuuteen vertaamalla perinteisiä kehitysmenetelmiä low-code-kehitykseen. Luvussa 5 analysoidaan perinteisillä kehitysmenetelmillä ja low-code-

kehityksellä tuotettujen järjestelmien välisiä laatueroja. Luvussa 6 käsitellään low-code-teknologiaan liittyviä riskejä sekä niiden hallintakeinoja. Luvussa 7 tehdään yhteenveto ja esitellään johtopäätökset.

2 Käsitteitä

Low-code-kehityksellä (Low-code Development, LCD, tai Low-code Programming, LCP) viitataan kehitysmenetelmiin, joissa luodaan, suunnitellaan ja kehitetään ohjelmistoja tai sovelluksia vähäisellä ohjelmakoodin kirjoittamisella tai kokonaan ilman sitä (Heras 2026). Low-code-kehitys tehostaa kehitysprosessin nopeutta, joustavuutta ja iteratiivisuutta muuttamalla kehityksen graafisessa käyttöliittymässä tapahtuvaksi visuaaliseksi prosessiksi (Rokis & Kirikova 2023). Low-code-teknologiaan viitataan myös käsitteellä Low-code/No-code, jossa ”No-code” viittaa tilanteeseen, jossa kehitysprosessissa ei vaadita lainkaan ohjelmakoodin kirjoittamista.

Low-code-kehitystä voi harjoittaa low-code-kehitysalustoilla, joihin viitataan yleisesti englanninkielisellä käsitteellä Low-code Development Platform (LCDP) tai jos kyseessä on määritelmän mukaisesti no-code-kehitys, voidaan käyttää nimitystä No-code Development Platform (NCDP) (Kedziora ym. 2024). Yleisimmin käytettyjä low-code-kehitysalustoja ovat muun muassa OutSystems, Mendix ja Microsoft PowerApps (Sahay ym. 2023) ja Salesforce Lightning (Viljoen ym. 2026).

Low-code-kehitysalustat tarjoavat ominaisuuksia, kuten visuaalisten elementtien asettelemiseen perustuvia käyttöliittymiä sekä valmiita komponentteja ja malleja, joiden avulla käyttäjät voivat kehittää ja ottaa käyttöön räätälöityjä sovelluksia ilman ohjelmointiosaamista (Viljoen ym. 2026). Low-code-alustoja ei kuitenkaan tule sekoittaa muihin samankaltaisiin teknologioihin kuten työnkulkuautomaatioalustoihin (Workflow Automation Platform, WAT) tai robotiikkaprosessiautomaatioon (Robotic Process Automation, RPA), joiden avulla käyttäjä voi optimoida tai automatisoida prosessejaan, mutta joita ei kuitenkaan sellaisenaan käytetä itsenäisten ohjelmistojen tai sovellusten kaltaisten kokonaisuuksien kehittämiseen (Viljoen ym. 2026).

Tietojärjestelmä (Information System, IS) on yhdistelmä älykkäitä toimijoita, kuten ihmisiä, sekä prosesseja ja tietotekniikkaa, kuten laitteistoja, ohjelmistoja ja infrastruktuuria, jotka liittyvät tiedon keräämiseen, jakamiseen ja hyödyntämiseen organisaatiossa (Meneses & Varajão 2022). Tietojärjestelmien kehittäminen (Information Systems Development, ISD) on organisaation strategiaa tukevien tietojärjestelmien suunnittelua ja toteuttamista liiketoimintamalleja ja liiketoimintaprosesseja hyödyntämällä (Meneses & Varajão 2022). Tässä tutkielmassa tietojärjestelmien kehittämisellä tarkoitetaan ensisijaisesti organisaatiokontekstissa tapahtuvaa sovellus- ja järjestelmäkehitystä.

Ohjelmistokehitys (Software Engineering) tarkoittaa tieteellisen tiedon käytännön soveltamista suunniteltaessa ja toteutettaessa tietokoneohjelmia sekä laadittaessa niihin liittyvää

dokumentaatiota, jota tarvitaan ohjelmiston kehittämiseen, käyttämiseen ja ylläpitoon (Boehm 1976). Tässä tutkielmassa ohjelmistokehitystä, johon viitataan selkeyden vuoksi myös käsitteillä “perinteinen ohjelmistokehitys” ja “perinteiset kehitysmenetelmät”, käsitellään laajasti tietojärjestelmien kehittämisen näkökulmasta. Näin ollen ohjelmistokehitystä koskevia lähteitä ja määritelmiä tulkitaan organisaatioiden tietojärjestelmien suunnitteluun, toteutukseen ja ylläpitoon läheisesti liittyvinä ilmiöinä.

Pilvipalveluiden ja vastaavien teknologioiden yhteydessä käsite toimittajaloukku (Vendor Lock-in) tarkoittaa tilannetta, jossa asiakas on riippuvainen yhden palveluntarjoajan teknisestä toteutuksesta eikä voi helposti siirtyä toisen toimittajan palveluihin myöhemmin ilman merkittäviä kustannuksia, oikeudellisia rajoitteita tai teknisiä yhteensopimattomuuksia (Opara-Martins ym. 2016).

Kansalaisohjelmoijalla (Citizen Developer, CD) viitataan henkilöön, jolla ei ole muodollista ohjelmistokehitykseen liittyvää koulutusta, mutta joka kehittää omassa liiketoimintayksikössään sovelluksia, joilla ratkotaan kyseisen toimintayksikön ongelmia (Viljoen ym. 2026). Kansalaisohjelmoija voi siis olla minkä tahansa alan ammattilainen, joka voi low-code-alustaa käyttämällä osallistua tietojärjestelmäkehitysprosessiin.

Liiketoiminnan automaatiolla viitataan eri sidosryhmien ja lähteiden digitaalisiin järjestelmiin syöttämän datan automaattiseen käsittelyyn, jolla pyritään helpottamaan ja virtaviivaistamaan liiketoimia ja työnkulkua (Ajimati ym. 2025).

3 Teoreettinen viitekehys

Tietojärjestelmätieteessä on olemassa monia yleisesti käytettyjä malleja, joiden avulla voidaan arvioida tietojärjestelmäprojektien onnistumista. Niin ikään tietotekniikan alalle on vakiintunut kattavia standardointeja, joilla voidaan taata tietojärjestelmien ja ohjelmistojen laatu ja käyttökelpoisuus. Sen sijaan nimenomaan tietojärjestelmäkehitysprojektin tehokkuutta mittaavia, sellaisenaan sovellettavissa olevia malleja ei juuri tunneta. Jotta tietojärjestelmäkehitysprosessin tehokkuutta koskevaan tutkimuskysymykseen voitaisiin vastata mahdollisimman kokonaisvaltaisesti, tässä osiossa rakennetaan alalla yleisesti käytetyistä standardoinneista ja malleista synteesi, jonka avulla perinteisiä tietojärjestelmäkehitysmenetelmiä sekä low-code-kehitystä voidaan vertailla luotettavasti.

3.1 Capability Maturity Model Integration (CMMI)

CMMI (Capability Maturity Model Integration) on Software Engineering Institute (SEI) -organisaation luoma prosessikypsyysmalli, jonka tavoitteena on auttaa organisaatioita parantamaan ohjelmistotuotannon ja järjestelmien kehityksen laatua, ennustettavuutta ja tehokkuutta. Malli perustuu ajatukseen siitä, että organisaation suorituskyky ei riipu pelkästään yksittäisistä käytännöistä tai työkaluista vaan ennen kaikkea siitä, kuinka hyvin sen prosessit on määritelty, hallittu, mitattu ja jatkuvasti paranneltu (SEI 2002).

SEI:n julkaisemassa CMMI v1.1 -versiossa määritellään viisi kypsyystasoa (Maturity Levels), joiden avulla organisaation prosessien kehittyneisyyttä voidaan arvioida: Initial (alkutaso), Managed (hallittu taso), Defined (määritelty taso), Quantitatively Managed (määrällisesti hallittu taso) ja Optimizing (optimoiva taso).

Mallissa käsitellään myös niin sanottuja prosessialueita, jotka kuvaavat ohjelmistokehityksen keskeisiä toimintoja, kuten vaatimusten hallintaa, projektinhallintaa, laadunvarmistusta, konfiguraation hallintaa sekä prosessien mittaamista ja analysointia. Näiden avulla organisaatiot voivat arvioida ja kehittää käytäntöjään systemaattisesti kohti ennustettavaa, mitattavaa ja jatkuvasti kehittyvää ohjelmistotuotantoa. Malli ei keskity yksittäiseen teknologiaan tai ohjelmistotuotteeseen vaan organisaation toimintatapoihin ja niiden kypsyyteen.

CMMI v1.1 -mallia käytetään tämän tutkielman viitekehyksessä tietojärjestelmäkehitysprosessin tehokkuuden arvioinnin akateemisena perustana. Malli innoittaa tarkastelemaan tietojärjestelmäkehitysprosessien kustannustehokkuutta, resurssitehokkuutta sekä ylläpidettävyyttä ja skaalautuvuutta.

3.2 ISO-standardit

ISO (International Organization for Standardization) ja IEC (International Electrotechnical Commission) ovat kansainvälisiä standardointiorganisaatioita, jotka kehittävät ja julkaisevat ISO/IEC-standardeja. Näiden standardien tavoitteena on yhtenäistää käytäntöjä tuotteiden, palveluiden, prosessien sekä järjestelmien laadun, turvallisuuden ja luotettavuuden arvioinnissa ja kehittämisessä. Nämä standardit tarjoavat yleisesti hyväksyttyjä viitekehyksiä, joiden avulla voidaan edistää järjestelmien yhteensopivuutta, vertailtavuutta sekä johdonmukaista arviointia eri toimialoilla. Lisäksi ne tukevat systemaattista lähestymistapaa riskienhallintaan ja laadunvarmistukseen.

3.2.1 ISO/IEC 25010:2023

ISO/IEC 25010:2023 (ISO/IEC 2023) on teollisuudessa sekä akateemisessa kirjallisuudessa laajasti käytetty ohjelmistotuotteiden laatustandardi, joka tarjoaa kattavan viitekehyksen ohjelmiston laadun arviointiin ja nykyaikaisten kehityskäytäntöjen tukemiseen (Naqvi ym. 2025). Sitä hyödynnetään tässä tutkielmassa akateemisena perustana tietojärjestelmän laadun arviointiin sekä soveltuvilta osin myös tietojärjestelmäkehitysprosessin arviointiin.

ISO/IEC 25010:2023-standardiin sisältyy noin kymmenkunta eri mitattavaa ominaisuutta, joista low-code-kontekstissa olennaisimpia ovat muun muassa toiminnallinen soveltuvuus, suorituskyky, yhteensopivuus, vuorovaikutuskyky, turvallisuus, ylläpidettävyys ja joustavuus. Toiminnallisella soveltuvuudella tarkoitetaan sitä, että järjestelmä on käyttäjäystävällinen ja vastaa todellisiin tarpeisiin. Suorituskyvyllä viitataan low-code-kontekstissa järjestelmän kykyyn vastata pyyntöihin nopeasti ajon aikana. Yhteensopivuudella tarkoitetaan sitä, että järjestelmä voi toimia olemassa olevien järjestelmien rinnalla sekä yhteistyössä niiden kanssa. Vuorovaikutuskyky mittaa käyttäjäystävällisyyttä. Turvallisuus tarkoittaa sitä, että järjestelmä on suojattu valtuuttamattomalta käytöltä. Ylläpidettävyys liittyy järjestelmän helppoon muunneltavuuteen, ja joustavuus tarkoittaa järjestelmän kykyä mukautua uusiin vaatimuksiin, käyttötapoihin ja toimintaympäristöihin. (Naqvi ym. 2025.)

CMMI:n tavoin ISO/IEC 25010:2023 -standardi innoittaa tarkastelemaan tietojärjestelmäkehitysprosessin tehokkuuden vertailussa ylläpidettävyyttä, mutta myös kehitysprosessin joustavuutta, nopeutta ja yhteensopivuutta. Järjestelmän laadun mittaamisessa se antaa aihetta tarkastella toiminnallista soveltuvuutta, ylläpidettävyyttä, suorituskykyä sekä siirrettävyyttä.

3.2.2 ISO 31000:2018

ISO 31000:2018 (ISO 2018) on laajasti käytetty kansainvälinen riskienhallinnan standardi, joka tarjoaa kokonaisvaltaisen lähestymistavan riskien tunnistamiseen, arviointiin ja käsittelyyn. Standardi määrittelee periaatteet ja prosessit, joiden avulla riskienhallinta voidaan liittää osaksi organisaation johtamista, päätöksentekoa ja operatiivista toimintaa. Sen tavoitteena on tukea johdonmukaista ja systemaattista epävarmuuden hallintaa sekä parantaa organisaation kykyä saavuttaa tavoitteensa muuttuvassa toimintaympäristössä.

Tässä tutkielmassa ISO 31000:2018-standardia käytetään soveltuvin osin viitekehyksenä low-code-kehitykseen liittyvien riskien arvioinnissa ja luokittelussa. Low-code-kehitykseen liitetyt riskit jäsennetään analyysin selkeyttämiseksi kolmeen pääluokkaan: teknisiin, hallinnollisiin sekä strategisiin ja liiketoiminnallisiin riskeihin.

3.3 Tutkielman vertailukriteerit

ISO-standardeja ja CMMI-mallia mukaillen low-code-kehityksen ja perinteisen tietojärjestelmäkehityksen tehokkuutta vertaillaan tässä tutkielmassa joustavuuden, kustannustehokkuuden, nopeuden, resurssitehokkuuden, ylläpidettävyyden ja skaalautuvuuden sekä yhteensopivuuden näkökulmista. Tietojärjestelmän laatua vertaillaan toiminnallisen soveltuvuuden, ylläpidettävyyden, suorituskyvyn sekä siirrettävyyden näkökulmista. Low-code-kehityksen riskejä tarkastellaan teknisten, hallinnollisten sekä strategisten ja liiketoiminnallisten riskien näkökulmista.

4 Miten low-code-alustat vaikuttavat tietojärjestelmäkehitysprosessin tehokkuuteen verrattuna perinteisiin tietojärjestelmäkehitysmenetelmiin?

4.1 Kehitysprosessin joustavuus

Perinteisen ohjelmistokehityksen piirteet, kuten ohjelmakoodin tarkka kirjoitusasu, etenemislogiikka ja muut muotoseikat ovat osasyy siihen, minkä takia ohjelman muuttaminen voi olla työlästä. Siinä missä käsin ohjelmoimalla pienikin muutostyö voi viedä huomattavasti aikaa, low-code-alustalla ohjelmaan voi helppokäyttöisen ja käyttäjäystävällisen käyttöliittymän ansiosta tehdä helposti muutoksia (Ajimati ym. 2025). Low-code-kehitys on tiimityöhön sopivaa, koska muutoksia voi nähdä reaaliajassa, jolloin tiimin jäsenten on helppoa tehdä yhteistyötä (Gao & Fagerholm 2026).

Low-code-alustat nopeuttavat kehitysprosessia ja laskevat kehitysprosessiin osallistumisen kynnystä ei-teknisille kehittäjille, mutta niiden varjopuoli on innovaatioiden ja räätälöinnin rajoittuminen alustan asettamalle tasolle (Naqvi ym. 2025). Useimmiten joudutaankin tyytymään ratkaisuun, joka on lähellä haluttua lopputulosta, mutta ei kuitenkaan täysin vastaa sitä. Rajallisuus liittyy siihen, että low-code-alustoilla ei välttämättä ole mahdollista tehdä pikkutarkkoja muutoksia pelkällä ohjelmakoodilla (Matos Claro ym. 2026).

Perinteinen käsin ohjelmoiminen mahdollistaa siis ohjelman kokonaisvaltaisen räätälöinnin ja hallinnan, mutta kehitysprosessiin ja ylläpitoon liittyy paljon työtä. Sen lisäksi, että kehittäjän täytyy huolehtia koodin kirjoitusasusta ja muista muodollisuuksista, hänen täytyy yleensä myös itse huolehtia kaikkien perustoiminnallisuuksien lisäämisestä. Esimerkiksi sovelluksen mobiili- ja selainkäyttäytyminen tulee usein ohjelmoida itse, toisin kuin low-code-alustoilla, jotka mukauttavat sovelluksen toiminnan kuhunkin ympäristöön sopivaksi automaattisesti. Myös tietosuojaominaisuuksista ja muista ohjelman toiminnan kannalta olennaisista seikoista perinteisen ohjelmistokehittäjän on huolehdittava itse. (Heras 2026.)

Low-code-kehityksen merkittävimpiin etuihin lukeutuva kehitysprosessin helppous voi kuitenkin olla subjektiivista: vaikka perusperiaate on, ettei raa'an ohjelmakoodin kirjoittamista tarvitse osata, kehittäjältä saatetaan silti edellyttää laajaa teknistä tuntemusta. Kandaurovan ja kumppaneiden (2024) tapaustutkimuksessa tekoälypohjaisten keskustelubottien luomiseen tähtäävässä low-code-

kehitysprojektissa käyttäjiltä vaadittiin vähintään perustason ohjelmointiosaamista sekä tietämystä algoritmeista ja koneoppimisen perusteista.

4.2 Kustannustehokkuus

Merkittävät kustannussäästöt ovat low-code-kehitykseen liitetty olettamus, joka ajaa organisaatioita omaksumaan tämän teknologian osaksi tietojärjestelmäkehitystään. Ajimatin ja kumppaneiden (2025) mukaan perinteisen ohjelmistokehityksen korkeat kustannukset selittyvät muun muassa sillä, että monimutkaisten ohjelmistojen kehittäminen edellyttää ohjelmistokehityksen ammattilaisista koostuvien tiimien muodostamista. On olemassa näyttöä siitä, että low-code-kehityksen kustannussäästöt selittyvät nimenomaan ohjelmistokehittäjien palkkakustannuksiin liittyvillä säästöillä (Ajimati ym. 2025).

Low-code-alustojen aikaansaamien palkkaussäästöjen hyötyjä voivat kuitenkin laskea alustoihin liittyvät lisensointimaksut. Perinteinen ohjelmistokehitys nojaa avoimen lähdekoodin kirjastoihin, joiden sisältämän koodin käyttäminen on yleensä maksutonta (Heras 2026). Perinteisessäkin ohjelmistokehityksessä voi toki olla käytössä maksullisia kehitystyökaluja, kuten koodikirjastoja tai tekstieditoreita, mutta näihin liittyvien kustannusten suuruusluokat ovat verrattain vähäpätöisiä.

4.3 Kehitysprosessin nopeus

Yksi low-code-kehityksen suurimpia valttikortteja on kehitysprosessin nopeus, joka selittyy muun muassa sillä, että ohjelman voi rakentaa visuaalisessa käyttöliittymässä ohjelmakoodia kirjoittamatta. Low-code-alustojen avulla mekaanisia prosesseja saadaan automatisoitua ja monimutkaisia tehtäviä yksinkertaistettua, mikä säästää myös kokeneiden ohjelmistokehittäjien aikaa (Fedeli ym. 2025). Lisäksi low-code-kehitys parantaa liiketoimintayksiköiden välistä yhteistyötä, mikä tekee kehityksestä ketterämpää (Fedeli ym. 2025). Low-code-kehityksen avulla ideat saadaan muutettua toimiviksi ratkaisuiksi paljon nopeammin kuin perinteisillä kehitysmenetelmillä, mikä on merkittävä kilpailuetu nykyajan kilpailullisessa ja nopeasti muuttuvassa liiketoimintaympäristössä (Ajimati ym. 2025).

Ajimatin ja kumppaneiden (2025) tutkimuksessa uusien ominaisuuksien lisäämiseen kului low-code-kehityksessä 30 % vähemmän aikaa kuin perinteisillä menetelmillä toteutettuna. Tämä selittyy sillä, että kansalaisohjelmoijat voivat low-code-alustojen avulla ratkoa liiketoiminnallisia ongelmia itsenäisesti. Low-code-alustojen valmiiden mallien ja ohjelmistolaajennuksien käyttö nopeuttaa kehitysprosessia entisestään. (Ajimati ym. 2025.) Valmiiden mallien käyttö näkyy myös kiinalaisten

sähköyritysten low-code-kehityksessä, jossa niitä hyödynnetään prosessien automatisoimisessa (Wei & Zhang 2025).

Myös Heras (2026) havaitsi low-code-kehityksen nopeuttavan kehitysprosessia merkittävästi. Low-code-alustalla toteutettu kognitiiviseen kuntoutukseen tarkoitettu järjestelmä saatiin valmiiksi noin 50 tunnissa, kun taas perinteisillä menetelmillä aikaa kului lähes 900 tuntia.

Helppokäyttöisyydellä on osansa low-code-alustojen nopeudessa, mutta alustojen omaksuminen on toisinaan myös haastavaa. Matos Claro ja kumppanit (2026) havaitsivat low-code-alustojen opettelun vievän paljon aikaa. Myös Kandaurova ja kumppanit (2024) havaitsivat alustan käyttöönoton vievän paljon aikaa tapaustutkimuksessaan, jonka kohteena olevassa projektissa sovelluksen laatu parani kokonaisuudessaan, mutta itse kehitysprosessi oli hankala, koska se vaati organisaation olemassa olevien prosessien kartoittamista ja standardoimista.

4.4 Resurssitehokkuus

Low-code-kehitystä omaksutaan myös laajan osaajapulan takia. Osaajapulaa ratkotaan muuttamalla kehitystyötä low-code-alustojen avulla sellaiseksi, että muutkin kuin ohjelmistokehityksen ammattilaiset voivat osallistua tietojärjestelmäkehitykseen. Näin myös ohjelmistokehityksen ammattilaisille jää enemmän aikaa keskittyä vaativien kehitys- ja asennustehtävien hoitamiseen, kun kansalaisohjelmoijat auttavat kehityksessä. Low-code-alustat myös parantavat tiimien välistä yhteistyötä. (Ajimati ym. 2025.)

Low-code-kehityksen tuomat resurssihyödyt nojaavat myös pitkälti mekaanisten prosessien automatisointiin. Koska perinteiseen ohjelmistokehitykseen kuuluvat ohjelmointi-, testaus- ja julkaisutoimenpiteet ynnä muut toiminnallisuuteen vaikuttavat vaiheet hoituvat low-code-alustoilla pitkälti automaattisesti, kehittäjät voivat keskittyä syvällisemmin uuden ominaisuuden kehittämiseen. Valmiita elementtejä voi tallentaa pilvitallennustilaan ja niitä voi käyttää uudelleen muissa projekteissa. (Ajimati ym. 2025.)

Ohjelmistokehityksen ammattilaiset eivät ole ainoa ammattiryhmä, joiden työtaakkaa low-code-alustojen avulla voidaan vähentää. Perinteisessä ohjelmistokehityksessä ohjelmistotuotteen suunnittelussa tarvitaan käyttöliittymäsuunnittelijoita niin suunnitteluvaiheessa kuin mahdollisesti myöhemmissä yksittäisissä muutoksissa. Koska low-code-alustalla käyttäjä näkee suoraan tekemiensä muutosten toiminnan ja ulkoasun, käyttöliittymäsuunnittelijoita ei välttämättä tarvita avuksi jokaisessa muutoksessa. (Gao & Fagerholm 2026.)

Low-code-alustojen käyttö on yleensä melko joustavaa ja helppoa, minkä ansiosta kehitystiimit voivat olla pieniä ja tiimejä voi myös sekoittaa tarpeen mukaan (Gao & Fagerholm 2026). Toisaalta low-code-kehitys on hyvin alustasidonnaista eli kokemus tietystä low-code-alustasta ei takaa, että kehittäjä osaisi suoraan käyttää jotain toista low-code-alustaa (Viljoen ym. 2026). Näin ollen low-code-kehitys voi siis olla este henkilöstöresurssien optimaaliselle käytölle, mikäli kehitysprojektien osaamisvaatimukset vaihtelevat. Perinteisessä ohjelmistokehityksessä tämä ongelma ei ole niin korostunut, koska esimerkiksi ohjelmointikielet toimivat samalla tavalla kehitysympäristöstä riippumatta (Viljoen ym. 2026).

4.5 Ylläpidettävyys ja skaalautuvuus

Low-code-alustoilla luotujen järjestelmien ylläpidosta on ristiriitaisia tuloksia. Kuten aiemmin mainittu, low-code-alustoilla on oletuksena ylläpitoa helpottavia ominaisuuksia. Ajimatin ja kumppaneiden (2025) mukaan low-code-alustalla toteutettuun ohjelmaan saadaan tarpeen mukaan lisättyä uusia ominaisuuksia nopeasti. Naqvin ja kumppaneiden (2025) mukaan sen sijaan low-code-kehityksessä skaalaus rajoittuu usein alustan ominaisuuksiin ja budjettirajoitteisiin, minkä lisäksi suuren mittakaavan muutokset olemassa olevaan järjestelmään ovat erittäin haasteellisia. Myös Kandaurova ja kumppanit (2024) havaitsivat tutkimuskohteensa tapauksessa, että olemassa olevan järjestelmän toimintojen automatisointi jälkikäteen vaati paljon ylläpitoa eikä toimintoja kyetty automatisoimaan halutulla tavalla.

Perinteisessä ohjelmistokehityksessä testaus on olennainen prosessi, jossa tarkastellaan koko järjestelmän toiminnallisuutta ja vuorovaikutuksia. Low-code-alustoilla testaus on yleensä suppeaa: huomion kohteena ovat pääasiassa järjestelmän tuottamat näkyvät tulosteet, minkä takia ohjelman sisäistä toimintaa ei välttämättä voida seurata tarkasti (Heras 2026). Jotkin low-code-alustat mahdollistavat melko kattavan testauksen, kun taas joillakin alustoilla testausominaisuudet ovat hyvin rajalliset, mikä pakottaa tekemään manuaalisia testejä tai käyttämään ulkoisia ratkaisuja (Naqvi ym. 2025). Myös versionhallinta, joka kuuluu olennaisesti perinteiseen ohjelmistokehitykseen, on joillakin low-code-alustoilla, kuten OutSystemsissä sekä Microsoft PowerAppsissa, puuttellista (Gao & Fagerholm 2026).

4.6 Yhteensopivuus

Koska tietojärjestelmät ovat monimutkaisia ja suuria kokonaisuuksia, jotka koostuvat useista eri ohjelmistoista, osien yhteensopivuus on ratkaisevaa. Ihannetapauksessa low-code-alustan tulisi sopia suoraan olemassa olevan järjestelmän päälle ilman suuria muutoksia. Kandaurovan ja

kumppaneiden (2024) tutkimuksessa low-code-alustan integroimista hankaloitti olemassa olevien järjestelmien dataformaattien yhteensopimattomuus kehitettävän järjestelmän kanssa, minkä lisäksi yhteensopivuusongelmia esiintyi järjestelmien rajapintojen kanssa.

Ongelmia low-code-alustojen implementoinnissa voi myös tuottaa alustan suunniteltu käyttötarkoitus. Siinä missä perinteisellä ohjelmoinnilla voidaan luoda järjestelmä minkä tahansa kokoiselle organisaatiolle, low-code-alustat on yleensä optimoitu suuryrityksille (Heras 2026). Vaikuttaa kuitenkin siltä, että etenkin pienet ja keskisuuret yritykset olisivat kiinnostuneita omaksumaan low-code-teknologiaa osaksi liiketoimintaprosessejaan (Gao & Fagerholm 2026).

5 Onko low-code-kehityksellä ja perinteisillä kehitysmenetelmillä tuotettujen järjestelmien laadussa eroja?

5.1 Toiminnallinen soveltuvuus

Low-code-alustat soveltuvat organisaation tavallisiin liiketoimintaprosesseihin, mutta suorituskyvyn osalta ne eivät välttämättä taivu monimutkaisiin tarpeisiin, kuten suurta laskentatehoa vaativiin tehtäviin tai suuren datamäärän käsittelyyn. Nopean luonteensa vuoksi low-code-alustat soveltuvat erinomaisesti nopeiden prototyyppien tekemiseen. Näin ollen organisaatiot, joiden on reagoitava nopeasti markkinoiden ja toimintaympäristön muutoksiin päivittämällä tietojärjestelmiään, voivat hyötyä low-code-alustoista. (Naqvi ym. 2025.)

Viime kädessä tietojärjestelmän laadun kuitenkin määrittää se, kuinka hyvin se vastaa organisaation tarpeita. Tietojärjestelmän tulisi olla helppokäyttöinen ja organisaation strategian mukainen kokonaisuus, joka tukee prosesseja eikä hankaloita niitä. Rajoitteista huolimatta low-code-kehityksessä muutokset voi nähdä reaaliajassa, minkä ansiosta palautetta on mahdollista saada heti alkuvaiheessa ja projektia voidaan alusta lähtien toteuttaa asiakaslähtöisesti (Gao & Fagerholm 2026).

5.2 Ylläpidettävyys

Kaikki järjestelmät vaativat elinkaarensa aikana monenlaista ylläpitoa, kuten päivittämistä, virheenkorjausta ja toiminnallisia muutoksia. Ylläpidon näkökulmasta järjestelmän laatuun vaikuttavat sekä ylläpitotoimenpiteiden määrä että ylläpidon helppous.

Koska tietojärjestelmät ovat perinteisesti isoja ja monimutkaisia kokonaisuuksia, ylläpidon tarve on jatkuvaa. Ohjelmakoodia tulee päivittää jatkuvasti, minkä lisäksi pitää seurata, etteivät päivitykset riko olemassa olevia toiminnallisuuksia. Jatkuvaa ylläpitotarvetta aiheuttavat muun muassa ohjelmointikieliin, kirjastoihin, käyttöjärjestelmiin sekä tietokantoihin liittyvät päivitykset. (Heras 2026.)

Low-code-alustat tarjotaan yleensä platform-as-a-service-palveluina (PaaS) eli palveluntarjoaja sisällyttää alustaan oletuksena kaikki tarpeelliset ominaisuudet. Näin ollen edellä mainittujen kaltaiset päivitykset tapahtuvat low-code-alustoilla yleensä automaattisesti. (Heras 2026.) Modulaarisen luonteensa vuoksi low-code-alustoilla luotuja järjestelmiä voi yleensä muuttaa ilman, että vaikutetaan tahattomasti joihinkin toiminnallisuuksiin. Vaikka rutiininomainen ylläpito on helppoa, räätälöityjen ominaisuuksien päivittäminen voi olla rajallista. (Naqvi ym. 2025.)

Viera-Gonzalezin ja kumppaneiden (2025) mukaan low-code-alustalla tuotetun järjestelmän ylläpitoa vaikeuttavat dokumentointimahdollisuuksien ja käyttöohjeiden puute. Kehittäjät eivät välttämättä ymmärrä alustojen toimintaa, jolloin he saattavat turvautua kolmansien osapuolten apuun, mikä osaltaan vähentää kehitysprosessin joustavuutta. Viera-Gonzalez ja kumppanit (2025) havaitsivat myös low-code-alustan päivitysten rikkovan sillä luodun järjestelmän toiminnallisuuksia, mikä pakottaa kehittämään joitain ominaisuuksia alusta uudella lähestymistavalla. Myös Liu ja kumppanit (2026) havaitsivat tutkimuksessaan samanlaista käytöstä low-code-alustoilla: kehittäjien ohjelmakoodilla räätälöimät ominaisuudet eivät enää toimineet päivitysten jälkeen.

5.3 Järjestelmän suorituskyky

Järjestelmän suorituskyvyllä tarkoitetaan esimerkiksi kykyä vastata käyttäjien pyyntöihin nopeasti ja tehokkaasti käytettävissä olevien resurssien puitteissa. Se koostuu muun muassa vasteajoista, resurssien käytöstä sekä kapasiteetista eli kyvystä skaalautua kasvavaan käyttäjä- ja datamäärään. Suorituskykyyn vaikuttavat muun muassa järjestelmän infrastruktuuri, koodin tehokkuus sekä kyky käsitellä suuria tietomääriä ja monimutkaisia kyselyjä ilman merkittäviä viiveitä. (Naqvi ym. 2025.)

Low-code-alustalla tuotettua järjestelmää ohjaava koodi ei välttämättä ole yhtä optimoitua kuin käsin kirjoitettu ohjelmakoodi, mikä voi johtaa korkeampaan järjestelmäresurssien käyttöön (Naqvi ym. 2025). Näin ollen low-code-alustoilla tuotettujen järjestelmien vasteaika voi olla heikko, mikäli niillä käytetään suuria datamääriä tai niiden täytyy käsitellä paljon liikennettä. Wei ja Zhang (2025) kuitenkin havaitsivat low-code-alustojen sisältävän kuormanhallintaominaisuuksia, jotka takaavat järjestelmän toimivuuden myös ruuhkaisina aikoina. Järjestelmät kykenevät heidän mukaansa myös tuottamaan reaaliaikaisia virheraportteja, mikä auttaa mahdollisten ongelmien korjaamisessa.

5.4 Siirrettävyys

Siirrettävyydellä tarkoitetaan sitä, että järjestelmä voidaan siirtää helposti toiseen ympäristöön. Koska low-code-alustat on yleensä rakennettu pilvialustojen päälle, low-code-kehitys voi saattaa organisaation toimittajaloukkuun (Naqvi ym. 2025). Vaikka perinteisessäkin tietojärjestelmäkehityksessä järjestelmien rakentaminen pilvialustoille on yleistä, avoimen lähdekoodin periaatteiden mukaisesti riippuvuus palveluntarjoajista ei välttämättä ole yhtä voimakas.

Liun ja kumppaneiden (2026) tutkimuksessa toimittajaloukku ilmeni siten, että low-code-alustan vaihtaminen oli monimutkaista alustojen eriävien datanhallintamenetelmien vuoksi. Eri alustoilla

saatettiin vaatia erilaisia dataformaatteja, ja myös datan käsittelystä vastaavat kyselykielet saattoivat olla erilaisia.

6 Mitä riskejä low-code-kehitykseen liittyy ja miten niitä voidaan hallita?

6.1 Tekniset riskit

Low-code-alustat eivät ole standardoituja, vaan ne ovat keskenään hyvin erilaisia. Ne ovat myös tietoturvan puolesta pirstaloituneita: eri alustoilla on eri tietoturvaominaisuuksia, jotka soveltuvat erilaisiin käyttötarkoituksiin (Ajimati ym. 2025). Koska low-code-alustat ovat yleensä kolmannen osapuolen tarjoamia, niitä käyttävän organisaation tulee luottaa palveluntarjoajan tietoturvakäytäntöihin eli siihen, että alusta on rakennettu tietoturvallisesti ja parhaita käytäntöjä noudattaen. Koska low-code-alustojen toiminta ei ole asiakasorganisaatioille täysin läpinäkyvää, niiden käyttö voi luoda tietoturvaan ja yksityisyydensuojaan liittyviä riskejä. Tämä pätee etenkin silloin, kun tietojärjestelmäkehitystä hoitavat kansalaisohjelmoijat, jotka eivät ole tietoisia kaikista tietosuojakäytännöistä. (Ajimati ym. 2025.)

Low-code-alustojen tietoturvattomuudesta on saatu viitteitä esimerkiksi datan validointiin liittyvien virheiden ja datan korruption muodossa (Naqvi ym. 2025; Wei & Zhang 2025). Toisin sanoen organisaation datan eheys voi kärsiä low-code-alustan implementoinnin seurauksena, mikä on merkittävä riski.

Käyttäjien toiminnan valvominen low-code-alustoilla voi olla haastavaa puutteellisten tietoturvaominaisuuksien takia. Kansalaisohjelmoijat saattavat kehittää tietämättään ja tietohallinnon huomaamatta haavoittuvaisen järjestelmän, joka pahimmassa tapauksessa julkaistaan ilman asianmukaista tarkistusta ja korjaamista (Ajimati ym. 2025). Viljoen ja kumppanit (2026) havaitsivat tutkimuksessaan kolme tapaa, joilla kansalaisohjelmointi pahentaa pimentoon jäävän kehityksen vaikutuksia: kansalaisohjelmoijat saattavat tehdä muutoksia ajossa oleviin ympäristöihin, low-code-alustojen omat ominaisuudet saattavat estää kansalaisohjelmoijien kehitysprosessin valvomisen, ja kansalaisohjelmoijat saattavat myös luoda low-code-alustoilla ohjelmistokehityksen ammattilaisille päällekkäisiä tukipyyntöjä, mikä rasittaa heitä ja vähentää organisaation resurssitehokkuutta.

Wein ja Zhangin (2025) tutkimus antaa low-code-alustojen tietoturvallisuudesta hyvin erilaisen kuvan kuin muut tässä tutkielmassa käytetyt tutkimukset. He huomasivat sähköverkon datanhallintaan rakennetun järjestelmän kehittämisen yhteydessä low-code-alustojen sisältävän

tietoturvaominaisuuksia, kuten datan salausta ja suojausta sekä pääsynhallintaa, joiden havaittiin parantavan järjestelmän tietoturvaa.

Perinteisessä ohjelmistokehityksessä olennainen osa virheenkorjausta ja tietoturvasta huolehtimista on lokitietojen automaattinen kerääminen eli jokaisen käyttäjän jokaisen liikkeen tallentaminen. Low-code-alustoilla näin tarkka seuraaminen ei yleensä ole mahdollista, minkä takia kehittäjät voivat tehdä tietämättään päällekkäisiä muutoksia ja ratkoa samoja asioita, ja tällaisten virhetilanteiden korjaaminen voi olla vaikeaa. (Viljoen ym. 2026.) Wei ja Zhang (2026) kuitenkin havaitsivat käyttämiensä low-code-alustojen sisältävän seurantatoimintoja käyttäjien tarkkaan valvontaan.

Tarkan dokumentaation ja toimintojen seuraamisen ansiosta perinteisessä ohjelmistokehityksessä myös järjestelmän ylläpito on yleensä melko suoraviivaista. Koska ohjelmakoodi kirjoitetaan tiettyä kaavaa noudattaen ja dokumentoidaan huolellisesti, periaatteessa kuka tahansa voi huolehtia minkä tahansa ominaisuuden korjaamisesta. Low-code-kehityksen tapauksessa voi käydä niin, että ainoa henkilö, joka pystyy päivittämään tiettyä ominaisuutta, on kyseisen ominaisuuden luoja, koska läpinäkymättömyyden takia muut kehittäjät eivät välttämättä ymmärrä ominaisuuden toimintaa. (Viljoen ym. 2026.) Näin ollen on olemassa riski merkittävän teknovelan syntymiselle, mikäli järjestelmän olennaisen osan kehittäjä vaikkapa poistuu organisaatiosta.

6.2 Hallinnolliset riskit

Organisaatiossa ei välttämättä ole selkeitä low-code-kehitykseen liittyviä kehityskäytäntöjä tai viestintää, joka edesauttaisi tietohallinnon ja kehittäjien välistä yhteistyötä (Ajimati ym. 2025). Tämä voi tehdä kehityksestä tehotonta ja jopa turhaa. Pahimmassa tapauksessa koko järjestelmä kehitetään väärän ongelman ratkaisemiseksi tai siitä tulee puutteellisen tietoturvan takia käyttökelvoton.

Uutena, organisaation vastuita ja rooleja muuttavana teknologiana low-code-kehitys voi myös kohdata vastustusta ja aiheuttaa konflikteja ilman oikeanlaista muutosjohtamista. Koska low-code-alustat antavat kansalaisohjelmoijille riippumattomuutta IT-ammattilaisista, näiden ryhmien välillä voi syntyä vastuualueisiin ja valtasuhteisiin liittyviä konflikteja (Ajimati ym. 2025). Organisaatiokulttuuri voi suosia tiettyjä ryhmiä, kuten liiketoiminnan ammattilaisia ja ohjelmistokehittäjiä, mikä voi asettaa muut työntekijät epämukavaan asemaan. Kommunikaatio ja yhteistyö voivat heikentyä, mikä myös heijastuu viime kädessä koko kehitysprojektin laatuun. (Ajimati ym. 2025.)

6.3 Strategiset ja liiketoiminnalliset riskit

Yksi merkittävimmistä strategisista low-code-kehitykseen liittyvistä riskeistä on toimittajaloukku (Matos Claro ym. 2026). Tämä on pahimmillaan kestämätön tilanne, jossa ollaan vastentahtoisesti riippuvaisia jostain tietystä low-code-alustasta, eikä järjestelmän siirtäminen toiselle palveluntarjoajalle ole mahdollista ilman merkittäviä kustannuksia.

Vaikka perinteisessäkin tietojärjestelmäkehityksessä on tavallista käyttää monien eri palveluntarjoajien alustoja, low-code-kehityksessä toimittajaloukkuun liittyvä riski on vähäisen standardoinnin takia korostunut. Toimittajaloukku voi tehdä low-code-investoinneista turhia, mikäli järjestelmä ei vastaa tarpeita eikä sitä voida kriittisellä hetkellä vaihtaa (Ajimati ym. 2025).

6.4 Hallintakeinoja

Low-code-kehityksessä on perinteiseen tietojärjestelmäkehitykseen verrattuna monia huomionarvoisia ominaispiirteitä, jotka tulisi ottaa huomioon jo ennen low-code-kehityksen aloittamista. Kunnollisessa low-code-kehityksen implementointistrategiassa tulisi tunnistaa niin low-coden tarjoamat hyödyt kuin siihen liittyvät haasteet. Olennaista on myös luoda low-coden käyttöönoton kannalta suotuisa organisaatiokulttuuri.

Kandaurova ja kumppanit (2024) rakentavat tutkimuksessaan viitekehyksen low-code-alustojen soveltuvuuden arviointiin sekä menestyksekkääseen käyttöönottoon, joka nojaa ennen varsinaista implementointia tapahtuvaan suunnittelu- ja valmistelutyöhön. Viitekehyksen pääkohdat on selitetty alla.

Ennen kuin potentiaalisia alustoja aletaan vertailla, on olennaista analysoida olemassa olevia liiketoimintaprosesseja riippuvuussuhteiden ja yhtenäisyyden kartoittamiseksi. Low-code-alustan käyttämiseen tarvittavan laaja-alaisen osaamisen saatavuudesta on varmistuttava. Yhteistyö toimintayksiköiden välillä tulee olla sujuvaa. (Kandaurova ym. 2024.)

Organisaation olemassa olevan infrastruktuurin tulee olla low-code-alustan implementoinnille suotuisa. Liiketoimintaprosesseja ja tietorakenteita tulee yhdenmukaistaa ja mukauttaa alustan vaatimuksiin sopiviksi. Ennen low-code-alustan ostamista sen ominaisuuksia ja kyvykkyyksiä tulee testata ja arvioida kattavasti. Alustan hankkimisen jälkeen tulee järjestää sen asianmukaiseen käyttöön opastavia koulutusohjelmia. (Kandaurova ym. 2024.)

On hyvä muistaa, ettei organisaatiokulttuurin tarvitse jämähtää käsitykseen, jonka mukaan henkilöstö koostuu vain liiketoiminnan ja ohjelmistokehityksen ammattilaisista sekä

kansalaisohjelmoijista. Todellisuudessa osaamistasoissa on eroja, ja kyvykkäimmissä kansalaisohjelmoijissa on potentiaalia toimia tietohallinnon ja muun organisaation välikäsinä, jotka auttaisivat low-code-kehityksen valvomisessa. Tällä voitaisiin ratkoa muun muassa kehitysprosessien päällekkäisyyteen ja parhaiden käytäntöjen noudattamiseen liittyviä ongelmia. (Viljoen ym. 2026.)

7 Yhteenveto ja johtopäätökset

Tässä tutkielmassa koitettiin selvittää, mitä mahdollisuuksia ja uhkia low-code-kehitys tuo tietojärjestelmäkehitykseen. Tämä toteutettiin vertaamalla low-code-kehitystä perinteiseen ohjelmistokehitykseen seuraavien tutkimuskysymysten avulla:

- Miten low-code-alustat vaikuttavat tietojärjestelmäkehitysprosessin tehokkuuteen verrattuna perinteisiin tietojärjestelmäkehitysmenetelmiin?
- Onko low-code-kehityksellä ja perinteisillä kehitysmenetelmillä tuotettujen järjestelmien laadussa eroja?
- Mitä riskejä low-code-kehitykseen liittyy ja miten niitä voidaan hallita?

Tutkimuskysymyksiin vastattiin vertailemalla kehitysprosessin ja tietojärjestelmän ominaisuuksia, joihin saatiin innoitus yleisesti käytetyistä ISO-standardeista ja CMMI-mallista. Ensimmäiseen tutkimuskysymykseen vastattiin vertailemalla kuutta erilaista kehitysprosessin ominaisuutta: joustavuutta, kustannustehokkuutta, nopeutta, resurssitehokkuutta, ylläpidettävyyttä ja skaalautuvuutta sekä yhteensopivuutta. Toiseen tutkimuskysymykseen vastattiin vertailemalla tietojärjestelmän laatua toiminnallisen soveltuvuuden, ylläpidettävyyden, järjestelmän suorituskyvyn sekä siirrettävyyden näkökulmista. Kolmanteen tutkimuskysymykseen vastattiin jakamalla low-code-kehityksen riskit teknisiin, hallinnollisiin sekä strategisiin ja liiketoiminnallisiin riskeihin, minkä lisäksi kartoitettiin yleisimpiä riskinhallintakeinoja.

7.1 Yhteenveto

Low-code-alustat voivat parantaa tietojärjestelmäkehitysprosessin tehokkuutta nopeuttamalla kehitystä ja vahvistamalla yhteistyötä, mutta tuotetun järjestelmän skaalaaminen ja pikkutarkka räätälöinti on haasteellista. Low-code-kehityksessä ideat saadaan muutettua toimiviksi ratkaisuiksi nopeammin kuin perinteisessä tietojärjestelmäkehityksessä. Low-code-alustojen ansiosta kansalaisohjelmoijat voivat kehittää järjestelmää itsenäisesti, mikä parantaa organisaation resurssitehokkuutta ja säästää palkkakustannuksia. Vaikka low-code-kehitys yksinkertaistaa kehitysprosessia, alustojen käytön opettelu voi olla aikaa vievää. Low-code-alustojen avulla voidaan saavuttaa kustannussäästöjä, jotka kuitenkin voivat huveta alustojen kalliisiin lisensointimaksuihin.

Low-code-kehityksellä tuotetun järjestelmän laatu voi olla perinteisellä tietojärjestelmäkehityksellä kehitettyä järjestelmää korkeampi, mutta todennäköisesti se on heikompi. Suuri laskentateho ja

suurien datamäärien käsittely edellyttää yleensä perinteistä tietojärjestelmäkehitystä, mutta low-code-alustat voivat olla hyviä työkaluja yksinkertaisten järjestelmien nopeassa kehittämisessä. Low-code-alustoilla tuotettujen järjestelmien ylläpito voi olla haastavaa puutteellisten dokumentointi- ja testausominaisuuksien vuoksi, minkä lisäksi alustojen päivitykset voivat rikkoa järjestelmän olemassa olevia toiminnallisuuksia. Koska low-code-alustat eivät toimi standardoiduilla tavoilla, alustan avulla tuotetun järjestelmän siirtäminen toiseen ympäristöön ei välttämättä ole mahdollista.

Yksi low-code-kehityksen suurimmista heikkouksista on puutteellisista valvonta- ja hallintaominaisuuksista johtuva huono tietoturva. Koska alustojen koodi ei ole läpinäkyvää, joudutaan luottamaan palveluntarjoajan tietoturvakäytäntöihin. Loppupeleissä kokemattomat kansalaisohjelmoijat ovat kuitenkin suurin tietoturvariski, sillä he voivat tietämättään ja tietohallinnon huomaamatta kehittää haavoittuvaisia järjestelmiä. Teknovelka, epäselvät kehityskäytännöt, heikko viestintä, organisaation sisäiset konfliktit sekä vahva toimittajaloukku ovat myös keskeisiä low-code-kehitykseen liittyviä riskejä. Riskejä voidaan hallita alustojen käyttöönottoa edeltävällä huolellisella suunnittelulla, liiketoimintaprosessien huolellisella kartoittamisella sekä muuntamalla olemassa oleva infrastruktuuri, prosessit ja tietorakenteet suotuisiksi low-code-alustan implementoinnille. Organisaation on varmistuttava laaja-alaisen osaamisen saatavuudesta sekä huolehdittava riittävästä käyttöönoton opastuksesta ja jatkokoulutuksesta.

7.2 Johtopäätökset

Low-code-kehitykseen liittyy paljon mahdollisuuksia, jotka voivat nopeuttaa tietojärjestelmäkehitystä, tehostaa resurssien käyttöä, laskea kustannuksia, helpottaa tietojärjestelmän ylläpitoa sekä parantaa järjestelmän laatua. Näihin mahdollisuuksiin liittyy myös paljon epävarmuustekijöitä, jotka tekevät perinteisestä tietojärjestelmäkehityksestä houkuttelevan vaihtoehdon.

Yksi low-code-kehityksen suurimmista heikkouksista on puutteellinen tietoturva, mikä on nykyisessä datavetoisessa liiketoiminnassa vakavasti otettava riskitekijä. Täytyy kuitenkin muistaa, että myös perinteiseen tietojärjestelmäkehitykseen liittyy samoja tietoturvariskejä kuin low-code-kehitykseen. Perinteisessä tietojärjestelmäkehityksessä on tavallista integroida monen eri palveluntarjoajien alustoja ja teknologioita samaan tietojärjestelmään, joten kolmannen osapuolen tietoturvakäytäntöihin liittyvä riski on yhtä lailla läsnä. Vaikka ohjelmistokehityksen ammattilaisilla on usein kattavat tiedot parhaista ja tietoturvallisimmista käytännöistä, kehittäjien osaamistaso vaihtelee ja järjestelmän haavoittuvuudet voivat jäädä yhtä lailla pimentoon kummassakin kehitystavassa. Low-code-kehitys ei ole yksiselitteisesti tietoturvattomampi vaihtoehto, sillä loppupeleissä käyttäjät ovat

suurin tietoturvariski, jonka hallintaan tarvitaan teknisten ominaisuuksien ohella asianmukaisia tietoturvajohtamista.

Organisaation kilpailukyvyn kannalta low-code-kehityksen lupaus nopeudesta ja resurssitehokkuudesta on houkutteleva, mutta huono siirrettävyys on merkittävä liiketoiminnan jatkuvuutta uhkaava riskitekijä. Koska siirrettävyysongelma ja toimittajaloukku kumpuavat usein datan yhteensopimattomuudesta, järjestelmän skaalaus- ja räätälöintirajoitteista sekä liian alhaisesta suorituskyvystä, low-coden menestyksekkääseen käyttöönottoon voisi riittää keskittyminen riittävän yksinkertaiseen käyttötarkoitukseen. Monimutkaisten, suuren mittakaavan järjestelmien luomiseen perinteinen tietojärjestelmäkehitys on potentiaalisin vaihtoehto, mutta pienissä ja yksinkertaisissa projekteissa, kuten organisaation sisäisten prosessien automatisoinnissa tai prototyyppien kehittämisessä, low-code-kehityksellä voidaan saada aikaan toimiva järjestelmä nopeasti.

Vaikka perinteisessä kehityksessä on low-code-kehitykseen verrattuna manuaalista työtä yleensä enemmän niin kehityksessä kuin ylläpidossa, low-code-kehityksessä mahdolliset järjestelmän uudelleenohjelmoinnit, ulkoiset konsultaatiot sekä alustojen käytön opettelu voivat kuroa eroa umpeen. Low-code-alustojen tietojärjestelmäkehitysprosessia nopeuttavasta vaikutuksesta on tähän mennessä näyttöä enimmäkseen yksittäisten tapaustutkimusten muodossa, joten luotettavan vertailun tekemiseksi tarvitaan laajempaa pitkittäistutkimusta. Niin ikään low-code-kehityksen vaikutuksesta tietojärjestelmän laatuun sekä kehityksen kustannuksiin tarvitaan lisää pitkän aikavälin tutkimusta.

Vaikka Low-code-kehitys ei nykymuodossaan täysin korvaa perinteistä tietojärjestelmäkehitystä, se voi oikein käytettynä tarjota enemmän mahdollisuuksia kuin uhkia. Parhaimmillaan se osallistaa organisaation kaikki työntekijät vahvemmin tietojärjestelmäkehitysprosessiin ja nopeuttaa ideointia sekä ideoiden muuntamista käyttökelpoisiksi ratkaisuiksi. Low-code-alustat kehittyvät koko ajan paremmiksi tekoälyn ja koneoppimisen kaltaisten teknologioiden vaikutuksesta (Naqvi ym. 2025) ja ovat jo tähän mennessä osoittautuneet arvokkaiksi, kilpailukykyä parantaviksi työkaluiksi monenlaisissa organisaatioissa. Ne eivät sovellu monimutkaisten, suuren mittakaavan sovellusten kehittämiseen, mutta yksinkertaisten prosessien kehittämiseen ne voivat olla erinomainen lisä perinteisen tietojärjestelmäkehityksen ohessa. Vaikka jotkut ohjelmistokehityksen ammattilaiset ovat epävarmoja low-code-kehityksen tekoälyintegraatioiden lisääntymisestä ja niiden vaikutuksista, low-code-kehityksen tulevaisuuden suhteen ollaan ammattipiireissäkin pääosin toiveikkaita (Gao & Fagerholm 2026). Haasteista huolimatta low-code-kehitystä ja perinteistä tietojärjestelmäkehitystä on

mahdollista tehdä menestyksekkäästi rinnakkain suotuisan organisaatiokulttuurin, asianmukaisen viestinnän ja strategisen suunnittelun avulla.

Lähteet

Ajimati, M. O., Carroll, N., & Maher, M. (2025). Adoption of low-code and no-code development: A systematic literature review and future research agenda. *Journal of Systems and Software*, *222*. https://doi.org/10.1016/j.jss.2024.112300

Boehm. (1976). Software Engineering. *IEEE Transactions on Computers*, *C–25*(12), 1226–1241. https://doi.org/10.1109/TC.1976.1674590

Fedeli, A., Di Salle, A., Micucci, D., Rebelo, L., Rossi, M. T., Mariani, L., & Iovino, L. (2025). How low-code platforms support digital twins of processes. *Software and Systems Modeling*, *24*(5), 1317–1333. https://doi.org/10.1007/s10270-025-01310-4

Gao, D., & Fagerholm, F. (2026). *An Investigation of Low-Code Development Adoption in a Finnish IT Consulting Firm*. *16361 LNCS*, 169–185. https://doi.org/10.1007/978-3-032-12089-2_11

Heras, F. (2026). *Traditional Coding, Low-Code and AI Assistants*. *2666 CCIS*, 183–198. https://doi.org/10.1007/978-3-032-08614-3_11

ISO/IEC, ISO/IEC 25010:2023, Systems and Software Engineering —Systems and Software Quality Requirements and Evaluation (SQuaRE)—Product Quality Model, 2nd ed. Geneva, Switzerland: International Organization for Standardization, 2023.

ISO/IEC, ISO 31000:2018, Risk management—Guidelines, 2nd ed. Geneva, Switzerland: International Organization for Standardization, 2018.

Kandaurova, M., Skog, D. A., & Bosch-Sijtsema, P. M. (2024). The Promise and Perils of Low-Code AI Platforms. *MIS Quarterly Executive*, *23*(3), 275–289. https://doi.org/10.17705/2msqe.00098

Kedziora, D., Aunimo, L., & Kortesalmi, H. (2024). *Between No-Code, Low-Code, Custom-Code, and AI. Hyperautomation Strategies at Nordic Enterprises*. *1076 LNNS*, 172–179. https://doi.org/10.1007/978-3-031-66594-3_18

Liu, Y., Chen, J., Bi, T., Grundy, J., Wang, Y., Yu, J., Chen, T., Tang, Y., & Zheng, Z. (2026). An empirical study on low-code programming using traditional vs large language model support. *Journal of Systems and Software*, *234*. https://doi.org/10.1016/j.jss.2025.112727

Matos Claro, S. M., Ruiz de la Peña, J., Serral Assension, E., & Snoeck, M. (2026). *MDE and LCNC Tools in the Generation of User Interfaces for Different Contexts of Use: A Systematic Literature Review*. *570 LNBIP*, 227–244. https://doi.org/10.1007/978-3-032-12063-2_15

Meneses, B., & Varajão, J. (2022). A framework of information systems development concepts. *Business Systems Research*, *13*(1), 84–103. https://doi.org/10.2478/bsrj-2022-0006

Naqvi, B., Kedziora, D., Zhang, L., & Oyedeji, S. (2025). Quality of Low-Code/No-Code Development Platforms Through the Lens of ISO 25010:2023. *Computer*, *58*(3), 30–40. https://doi.org/10.1109/MC.2024.3493192

Opara-Martins, J., Sahandi, R., & Tian, F. (2016). Critical analysis of vendor lock-in and its impact on cloud computing migration: A business perspective. *Journal of Cloud Computing*, *5*(1), 4. https://doi.org/10.1186/s13677-016-0054-z

Rokis, K., & Kirikova, M. (2023). Exploring Low-Code Development: A Comprehensive Literature Review. *Complex Systems Informatics and Modeling Quarterly*, *2023*(36), 68–86. https://doi.org/10.7250/csimq.2023-36.04

Sahay, A., Di Ruscio, D., Iovino, L., & Pierantonio, A. (2023). Analyzing business process management capabilities of low-code development platforms. *Software - Practice and Experience*, *53*(4), 1036–1060. https://doi.org/10.1002/spe.3177

SEI, 2002. Capability Maturity Model Integration (CMMISM), Version 1.1. SEI, CMU/SEI-2002-TR-029.

Shi, Z., Dong, J., & Gan, Y. (2025). Democratizing Digital Transformation: A Multisector Study of Low-Code Adoption Patterns, Limitations, and Emerging Paradigms. *Applied Sciences (Switzerland)*, *15*(12). https://doi.org/10.3390/app15126481

Viera-Gonzalez, P. M., Gonzalez-Gonzalez, O. H., & Sanchez-Guerrero, G. E. (2025). Low-Code Automation for Resolving Incidents during Class Schedule Registration. *Computer*, *58*(3), 41–48. https://doi.org/10.1109/MC.2024.3510463

Viljoen, A., Stelzl, B., Yang, M., Nguyen, J., Hein, A., Elshan, E., & Krcmar, H. (2026). Navigating Flexibility and Standardisation in Low-Code/No-Code Development. *Information Systems Journal*, *36*(1), 95–109. https://doi.org/10.1111/isj.70001

Wei, W., & Zhang, J. (2025). *Automation and Data Aggregation Algorithm for Low Code Development in Power Grid Mobile Report Generation*. *1351 LNNS*, 70–80. https://doi.org/10.1007/978-3-031-88287-6_7

Liitteet

Liite 1 Selvitys tekoälyn käytöstä

Tämän kandidaatintutkielman laatimisen tukena on käytetty Microsoft Copilotia (versio 2/2026). Työkalua on hyödynnetty tutkimusaineiston hakemisessa sekä potentiaalisten tutkimusartikkeleiden sisältöjen tiivistämisessä alustavan tarkastelun helpottamiseksi.

Käytetylle työkalulle on syötetty muun muassa seuraavanlaisia kehotteita:

"Hello! I need to find an academic definition for information system development from a reviewed academic source. Can you help me find such sources?"

"Hello! I will attach a research article. I would like you to analyse it thoroughly from an information systems manager's point of view. I want to know what the research is about and what the findings are. Please answer briefly."

Aiheen, hakutermien ja tutkimuskysymysten valinta sekä aineistojen lopullinen tulkinta perustui viime kädessä akateemiseen tutkimuskirjallisuuteen sekä tutkielman laatijan omaan kriittiseen ajatteluun. Tämän tutkielman sisältö on tutkielman laatijan itse tuottamaa ja arvioimaa.