TURUN YLIOPISTO

Ohjelmistojen suunnittelu- ja kehitysprosessien vaikutus kestävän kehityksen toteutumiseen

Ohjelmistojen kädenjälkivaikutukset ohjelmistokehityksen käytänteissä

Tietoliikenne- ja kyberturvallisuusteknologia
Tieto- ja viestintätekniikan tutkinto-ohjelma
Tietotekniikan laitos, Teknillinen tiedekunta
Diplomityö

Laatija:
Mikko Tapani Perkkiö

Ohjaajat:
Apulaisprofessori Tuomas Mäkilä (Tietotekniikan Laitos)

Toukokuu 2026

Diplomityö
Tietotekniikan laitos, Teknillinen tiedekunta
Turun yliopisto

Oppiaine: Ohjelmistotekniikka
Tutkinto-ohjelma: Tietotekniikka (DI)
Tekijä: Mikko Tapani Perkkiö
Otsikko: Ohjelmistojen suunnittelu- ja kehitysprosessien vaikutus kestävän kehityksen toteutumiseen
Sivumäärä: 106 sivua
Päivämäärä: Toukokuu 2026

Tiivistelmä

Digitaaliset ratkaisut ovat keskeisessä roolissa yhteiskunnan toiminnassa, ja ohjelmistojen merkitys taloudessa, julkishallinnossa ja arjen prosesseissa kasvaa jatkuvasti. Yhdistyneiden kansakuntien kestävänkehityksen tavoitteet (SDG-tavoitteet) edellyttävät, että myös digitaalinen infrastruktuuri suunnitellaan ja toteutetaan vastuullisesti. Ohjelmistojen vaikutukset eivät kuitenkaan ole neutraaleja: huonosti suunnitellut järjestelmät voivat lisätä resurssien ylikäyttöä, kasvattaa negatiivista hiilijalanjälkeä sekä heikentää ohjelmistojen myönteistä hiilikädenjälkivaikutusta ja kiertotalouden toteutumista. Tästä huolimatta kestävyyteen liittyvät näkökulmat jäävät ohjelmistokehityksessä usein toissijaisiksi.

Akateemisesta näkökulmasta tutkimusongelma liittyy siihen, että ohjelmistokehityksen kestävyyttä koskeva kirjallisuus on hajanaista ja tarkastelee tyypillisesti vain yksittäisiä kestävyyden ulottuvuuksia, kuten energiatehokkuutta, elinkaaren aikaisia päästöjä tai eettisiä ja sosiaalisia kysymyksiä. Teoreettisia viitekehyksiä, kuten Karlskrona Manifesti ja Green Software Engineering -periaatteet, on esitetty, mutta niiden systemaattinen soveltaminen organisaatioiden ohjelmistokehitysprosesseihin on vielä puutteellista. Tutkimus on siiloutunutta, eikä ohjelmistojen käytön aikaisia kädenjälkivaikutuksia ja systeemisiä vaikutuksia ole tarkasteltu riittävästi kokonaisvaltaisesta elinkaarinäkökulmasta.

Tämän diplomityön tavoitteena on tarkastella, miten kestävän kehityksen ja erityisesti hiilikädenjälkivaikutusten näkökulmat voidaan integroida osaksi ohjelmistojen suunnittelu- ja kehitysprosesseja koko elinkaaren ajan. Tutkimuksessa tarkastellaan sekä huonosti suunniteltujen ohjelmistojen mahdollisia negatiivisia ympäristö- ja systeemivaikutuksia että keinoja, joilla ohjelmistokehittäjät ja organisaatiot voivat tunnistaa, arvioida ja hallita näitä vaikutuksia käytännössä.

Tutkimus toteutetaan laadullisena tapaustutkimuksellisena tutkimuksena, jossa yhdistetään teoreettinen kirjallisuuskatsaus ja empiirinen haastattelututkimus Suomessa toimivissa ohjelmistoalan organisaatioissa. Haastatteluaineiston sisällönanalyysin avulla analysoidaan, miten kestävyys näkyy ohjelmistojen teknisessä suunnittelussa, arkkitehtuurivalinnoissa, projektimalleissa, osaamisessa ja asiakasvaatimuksissa. Lisäksi työssä hyödynnetään kestävän ohjelmistokehityksen kirjallisuutta, ICT:n vaikutustasojen ja elinkaariajattelun malleja sekä kestävyyden arviointiin liittyviä viitekehyksiä.

Työn keskeinen kontribuutio on poikkitieteellisen viitekehyksen ja kestävyysmallin kehittäminen, joka yhdistää ohjelmistokehityksen tekniset, organisatoriset ja kestävyyteen liittyvät näkökulmat. Tutkimus tuottaa sekä akateemista ymmärrystä ohjelmistojen kestävyyden ilmiöstä että käytännönläheisen mallin ohjelmistokehitystiimeille ja organisaatioille, jotka pyrkivät integroimaan kestävyyden osaksi kehitysprosessejaan ja vastuullisuusraportointiaan.

Asiasanat: kestävä ohjelmistokehitys, hiilikädenjälki, ohjelmiston elinkaari, ei-toiminnalliset vaatimukset, systeemivaikutukset, green software engineering

Sisällysluettelo

1 Johdanto

Digitaaliset ratkaisut ovat yhä kiinteämpi osa yhteiskunnan toimintaa, organisaatioiden prosesseja ja ihmisten arkea. Ohjelmistojen merkitys ei rajoitu enää yksittäisiin teknisiin järjestelmiin, vaan ne vaikuttavat laajasti siihen, miten palveluita tuotetaan, resursseja käytetään ja päätöksiä tehdään. Tämän vuoksi ohjelmistojen suunnittelu- ja kehitysratkaisuilla on vaikutuksia, jotka ulottuvat teknistä toteutusta laajemmalle. Kestävyyden näkökulmasta ohjelmisto ei ole neutraali ratkaisu, sillä sen vaatimukset, arkkitehtuuri, käyttö ja elinkaaren aikaiset päätökset voivat joko vähentää tai lisätä ympäristöön, talouteen ja yhteiskuntaan kohdistuvia vaikutuksia. Kestävän ohjelmistokehityksen merkitys on kasvanut erityisesti siksi, että digitaalisia järjestelmiä käytetään yhä laajemmin myös kestävyyshaasteiden ratkaisemiseen. Samalla on tunnistettava, että ohjelmistot voivat aiheuttaa myös kielteisiä vaikutuksia.

1.1 Tutkimuksen tausta

Ohjelmistokehityksessä kestävyyden käsite on edelleen uusi käsite, jota useimmat ohjelmistoyritykset ja -kehittäjät pyrkivät sisällyttämään ohjelmistojärjestelmiinsä. Ohjelmistoalalla tunnistetaan hyvin teknisen kestävyyden osa-alueita: tekninen velka tai taloudellinen kestävyys, kuten ohjelmiston kehittämisen ja ajamisen kustannukset. Ympäristöllinen kestävyys on herättänyt enemmän kiinnostusta viimeisinä vuosina. Euroopan Unioinin vuonna 2022 voimaan tullut CSRD-direktiivi (Corporate Sustainability Reporting Directive) ja International Sustainability Standards Board (ISSB) määrää, että jokaisen International Financial Reporting Standards (IFRS) -standardeja noudattavan yrityksen on raportoitava päästönsä, joihin luetaan mukaan myös ICT-järjestelmistä aiheutuvat päästöt. (IFRS Foundation, 2022.)

Ohjelmisto ei ole ympäristöystävällistä vain siksi, että se on virtuaalista. Ohjelmiston kehittämisessä, käyttöönotossa ja ylläpidossa käytetyillä menetelmillä ja prosesseilla on yhteiskuntaan ja ympäristöön kohdistuvia vaikutuksia, joita nykyiset ohjelmistokehityskäytännöt eivät yleensä huomioi (Albertao, 2010). Dick ja Naumann (2010) toteavat, että liiketoiminnallisesta näkökulmasta tarkasteltu ohjelmistokehityksen elinkaarimalli, ei sovellu ohjelmistojärjestelmien kestävyyteen kohdistuvien vaikutusten tunnistamiseen. Nykyisen mallin pääpaino on liiketoiminnassa sekä kehitys ja ylläpitovaiheissa ilman kestävyyden huomioimista. EU:n uudet kestävyyssääntelyt, kuten

CSRD, lisäävät organisaatioiden velvoitteita mitata ja raportoida ICT-järjestelmien ympäristövaikutuksia (EU, 2022).

Kehitettäessä sosio-teknisiä järjestelmiä on nostettava esiin jatkuvasti kaksi peruskysymystä: ”Rakennammeko oikean järjestelmän?” ja ”Rakennammeko järjestelmän oikein?”. Liiketoiminnassa tulkitaan ensimmäistä kysymystä usein suppeasti sen liiketoimintaongelman yhteydessä, johon ohjelmistojärjestelmän ratkaisulla pyritään. On hyvä kysyä, voidaanko järjestelmää, joka ei tue yhteiskuntamme kestävyyttä, kutsua ”oikeaksi järjestelmäksi”? Ohjelmistosuunnittelijoiden täytyy varmistaa, että ohjelmoijat tuottavat järjestelmän, joka edistää myös kestävyyttä.

Tutkimusyhteisö ei ole riittävästi paneutunut kestävyyden tarkasteluun ohjelmistojärjestelmien laatuattribuuttina. Ohjelmiston laadulla tarkoitetaan sitä astetta, jossa ohjelmisto sisältää halutun yhdistelmän ominaisuuksia. Se voidaan myös määritellä ja tarkastella kahdesta eri näkökulmasta. Ensimmäinen on vaatimustenmukaisuus määrittelyihin nähden, mikä tarkoittaa ohjelmiston laatua, jossa mitattavat ominaisuudet täyttävät ennalta määriteltyihin spesifikaatioihin perustuvat kiinteät vaatimukset. Toinen on asiakastarpeiden täyttäminen, mikä tarkoittaa ohjelmiston laadun kykyä vastata asiakkaiden tarpeisiin ja odotuksiin, olivat ne eksplisiittisiä tai implisiittisiä. (Berander ym., 2005.)

Ohjelmistokestävyyden tarkasteleminen ohjelmiston laadun näkökulmasta voi vaikuttaa olennaisesti ja myönteisesti ohjelmistojärjestelmien suunnittelu- ja kehitysprosesseihin, sillä laatuattribuutit kuuluvat järjestelmän ei-toiminnallisiin vaatimuksiin. Nämä vaatimukset kattavat kokonaisvaltaisesti ne tekijät, jotka ohjaavat järjestelmän ajonaikaista käyttäytymistä ja käyttäjäkokemusta, sekä edustavat huolenaiheita, joilla on potentiaalia laaja-alaiseen vaikutukseen järjestelmän eri arkkitehtuurikerroksissa (Microsoft, 2014).

Ohjelmistojärjestelmän kestävyyden tarkastelu mitattavana laatuattribuuttina perustuu useisiin ratkaiseviin edellytyksiin, kuten johdon ja henkilöstön riittävään ymmärrykseen, motivaatioon ja sitoutumiseen sekä taloudellisten, yhteiskunnallisten ja ekologisten ulottuvuuksien huomioon ottamiseen. Perinteiset ohjelmistokehitysprosessit eivät tällä hetkellä tue kestävyyden systemaattista integrointia, mikä johtaa siihen, että kestävyyttä koskevat toimenpiteet jäävät tehottomiksi tai kokonaan toteuttamatta (Penzenstadler, 2013) ja lopputuloksena on kestämättömiä ohjelmistoratkaisuja.

Nykyiset ohjelmistokehityksen elinkaarimallit eivät sisällä kestävyyden näkökulmaa, mikä on rajoittanut organisaatioiden ja ohjelmistokehittäjien mahdollisuuksia integroida kestävyys osaksi kehitysprosessia kestävien järjestelmien tuottamiseksi sekä niiden ympäristövaikutusten arvioimiseksi. Kyseiset vaikutukset voidaan jaotella ensimmäisen, toisen ja kolmannen asteen vaikutuksiin (Hilty & Aebischer, 2015).

Ohjelmisto kestävälle ihmiselle tarkoittaa ohjelmistoa, joka tukee käyttäjää kestävämpien valintojen ja toimintatapojen tekemisessä. Määritelmä ei kuitenkaan riittävästi erota kestävyyden ekologisia, taloudellisia ja sosiaalisia ulottuvuuksia ohjelmistojärjestelmien kontekstissa, minkä vuoksi käsitteen tarkempi määrittely myös laatuattribuuttina on tarpeellinen. Kestävä ohjelmistokehitys yhdistää periaatteet ja käytännöt siten, että ohjelmistot ovat teknisesti korkeatasoisia, tuottavat merkittävää liiketoiminta-arvoa ja pystyvät kehittymään nopeasti vastatakseen liiketoiminta- tai teknologisen toimintaympäristön muutoksiin (Tate, 2005). Jotta kestävyys voisi toteutua laatuattribuuttina, myös itse kehitysprosessin on oltava kestävä, sillä prosessin ominaisuudet vaikuttavat suoraan kehitettävien ohjelmistojen kestävyyteen. Kestävyyden tarkasteleminen ohjelmistojärjestelmien laatuattribuuttina ei rajoitu pelkästään energiatehokkuuteen, joka liittyy ohjelmistojen ensimmäisen asteen vaikutuksiin (Hilty & Aebischer, 2015). Ohjelmistokehityksessä kestävyys tulee huomioida ei-toiminnallisena vaatimuksena eli laatuattribuuttina samalla tavoin kuin esimerkiksi tietoturva, käytettävyys ja siirrettävyys, jotta myös toisen ja kolmannen asteen vaikutukset voidaan ottaa huomioon järjestelmäkontekstissa, vaikka niiden arviointi on haastavaa (Penzenstadler ym., 2014; Becker ym., 2015; Hilty & Aebischer, 2015). Tällainen lähestymistapa mahdollistaa ohjelmistojen kestävyyden merkittävän parantamisen taloudellisesta, sosiaalisesta ja ympäristöllisestä näkökulmasta (Naumann ym., 2011; Penzenstadler ym., 2014).

Ohjelmistojärjestelmät muodostavat keskeisen osan tieto- ja viestintäteknologiaa (ICT), ja niiden avulla voidaan lieventää ICT:n haitallisia ympäristövaikutuksia, mukaan lukien ensimmäisen, toisen ja kolmannen asteen vaikutukset. Kestävyyden näkökulmasta ICT:hen liittyy kaksi keskeistä osa-aluetta: Green by IT, joka tarkastelee sitä, miten kestävyyttä voidaan edistää ICT:n keinoin, sekä Green IT, joka keskittyy siihen, miten itse ICT:stä voidaan tehdä kestävämpää.

Kestävyyteen liittyvien vaatimusten käsittely sekä niiden järjestelmällinen tukeminen määrittelyn, analyysin ja toteutuksen osalta on ongelma, jota ei vielä ole ratkaistu.

Vuosikymmeniä sitten ohjelmistotekniikan ala kohtasi samankaltaisia puutteita prosesseissaan sisällyttämällä turvallisuuden ja tietoturvan uusiksi järjestelmän laatuominaisuuksiksi (Penzenstadler ym., 2014).

Kestävyyttä laatuominaisuutena ohjelmistotekniikan alalla on tutkittu hyvin vähän ja se tarjoaa selkeän tutkimusaukon, johon tämä diplomityö keskittyy. Tämä työ tarkastelee, miten ohjelmistokehityksessä voidaan siirtyä yksittäisistä energiatehokkuuden tai vihreän koodauksen näkökulmista kohti kokonaisvaltaista kestävyyden hallintaa. Tässä työssä huomioidaan ohjelmiston koko elinkaari, vaatimukset, arkkitehtuuri, projektinhallinta, organisaation vastuut, mittarit sekä ohjelmistojen myönteiset ja kielteiset systeemivaikutukset.

1.2 Tavoite

Tämän tutkimuksen tavoitteena on tarkastella, miten kestävän kehityksen näkökulmat voidaan integroida osaksi ohjelmistojen suunnittelu- ja kehitysprosesseja koko ohjelmiston elinkaaren ajan. Tutkimuksessa pyritään sekä kuvaamaan nykyisiä ohjelmistokehityksen käytäntöjä että rakentamaan poikkitieteellinen viitekehys, joka tukee ohjelmistokehittäjiä ja organisaatioita ohjelmistoprojektien kestävyyden arvioinnissa ja kehittämisessä.

Tutkimus toteutetaan kvalitatiivisena tutkimuksena yhdistämällä teoreettinen kirjallisuuskatsaus eri tieteenaloilta ja empiirinen haastattelututkimus Suomessa toimivissa ohjelmistoyrityksissä. Tavoitteena on ymmärtää, miten kestävyyteen liittyvät näkökulmat ilmenevät ohjelmistojen teknisessä suunnittelussa, arkkitehtuurivalinnoissa, projektimalleissa, tiimien osaamisessa sekä asiakkaiden asettamissa vaatimuksissa. Haastatteluaineisto analysoidaan sisällönanalyysin avulla. Näin tunnistetaan keskeisiä teemoja, puutteita ja kehityspotentiaalia ohjelmistokehityksessä.

1.3 Tutkimuskysymykset

Tutkimuskysymykset ovat seuraavat:

- TK1: Miten kestävän kehityksen ja kädenjälkivaikutusten näkökulmat voidaan integroida ohjelmistokehityksen eri vaiheisiin?
- TK2: Millaisia negatiivisia hiilijalanjälki- ja systeemivaikutuksia huonosti suunnitelluilla ohjelmistoilla voi olla?

- TK3: Miten ohjelmistokehittäjät ja organisaatiot voivat käytännössä tunnistaa, mitata ja hallita näitä vaikutuksia käytännössä?

- TK4: Voidaanko kehittää poikkitieteellinen viitekehys tai mittaristo, joka tukee ohjelmistojen kestävyyden ja kädenjälkivaikutusten systemaattista arviointia osana organisaatioiden ohjelmistokehitysprosesseja?

Työn keskeinen kontribuutio on kokonaisvaltaisen näkemyksen muodostaminen ohjelmistokehityksen kestävyydestä sekä empiirinen analyysi siitä, missä määrin kestävyyden periaatteet ovat siirtyneet teoriasta käytäntöön suomalaisissa ohjelmisto-organisaatioissa. Tutkimuksen tuloksena syntyvä viitekehys tarjoaa konkreettisen työkalun ohjelmistokehitystiimeille ja organisaatioille, jotka pyrkivät integroimaan kestävyyden osaksi kehitysprosessejaan, päätöksentekoaan ja vastuullisuusraportointiaan. Lisäksi työ syventää akateemista ymmärrystä ohjelmistojen kestävyyden ilmiöstä ja tukee digitaalisen yhteiskunnan vihreää siirtymää.

1.4 Tutkimusmenetelmät

Tutkimus toteutetaan laadullisena tapaustutkimuksellisena tutkimuksena. Tutkimuksen tavoitteena on ymmärtää, kuinka kestävyyden näkökulmat voidaan integroida ohjelmistokehityksen eri vaiheisiin sekä millaisia systeemisiä vaikutuksia ohjelmistosuunnittelulla syntyy. Laadullinen lähestymistapa valittiin tutkimuskohteena olevan ilmiön sosio-teknisen luonteen, kontekstisidonnaisuuden ja monitasoisuuden vuoksi, sillä sitä ei voida tarkastella pelkästään kvantitatiivisin mittarein (Eskola & Suoranta,1996 s. 61–62; Eriksson & Kovalainen, 2016 s. 4–5).

Aineisto kerätään tarkoituksenmukaisella otannalla valituilta asiantuntijoilta puolistrukturoitujen haastattelujen avulla sekä niitä täydentävistä dokumenttilähteistä. Analyysimenetelmänä käytetään temaattista analyysiä yhdistettynä laadulliseen sisältöanalyysiin, mikä mahdollistaa merkitysrakenteiden, mekanismien ja vaikutussuhteiden systemaattisen tunnistamisen aineistosta (Kiger ym., 2020). Analyysi toteutetaan yhdistämällä deduktiivinen lähestymistapa, joka perustuu teoreettiseen viitekehykseen, ja induktiivinen lähestymistapa, jossa tunnistetaan aineistosta esiin nousevia uusia teemoja. Menetelmälliset valinnat perustuvat tutkimuskysymysten luonteeseen ja noudattavat empiirisen ohjelmistotekniikan metodologisia suosituksia, joiden mukaan tutkimusstrategian tulee olla

johdonmukaisesti sidoksissa tutkimuksen tavoitteisiin ja tarkasteltavaan ilmiöön (Runeson & Höst, 2009; Wohlin ym., 2012).

1.5 Rajaus

Kestävän ohjelmiston kehittäminen on laaja-alainen aihe, jonka kehityksessä tulee huomioida kestävyyden kaikki kolme pilaria: sosiaalinen, taloudellinen ja ympäristöllinen. Alan kirjallisuus on hajanaista ja keskittyy usein vain yhteen kestävyyden ulottuvuuteen: ekologiseen tai taloudelliseen tai sosiaaliseen. Ohjelmistojen kehitysprosessissa tulisi huomioida toteutettavan ohjelmiston suunnittelun, kehitysprosessin ja toteutuksen kestävyys ja projektin tuotoksena syntyvän tuotteen kestävyys kaikilla kolmella osa-alueella.

Tämä diplomityö keskittyy selvittämään ohjelmiston suunnittelu- ja toteutusvaiheen vaikutuksia syntyvän lopputuotteen kestävyyteen. Työn tavoitteena on selvittää, mitkä suunnittelu ja toteutusvaiheen tekijät vaikuttavat ohjelmiston kestävyyteen sen käytön aikana. Ohjelmiston kestävyysvaikutukset ovat usein sidoksissa käyttötarkoitukseensa, joihin ohjelmoinnilla ei voida vaikuttaa.

Työn ulkopuolelle rajataan ohjelmistokehityksen suorat operatiiviset vaikutukset, joita ovat työstä, sen tekemisestä, työympäristöstä ja matkustamisesta aiheutuvat kestävyysvaikutukset. Diplomityö ottaa huomioon kestävyyden kaikki kolme pilaria, sosiaalisen, taloudellisen ja ympäristöllisen sekä ohjelmiston suunnittelussa että määrittelyssä. Ohjelmointiprosessissa on systemaattisesti huomioitava ohjelmiston koko elinkaaren aikainen hiilijalanjälki, jonka ohjelmisto synnyttää käytön aikana sekä virhetilanteissa. Työssä tarkastellaan myös ohjelmistojen mahdollisia systeemivaikutuksia. Näitä vaikutuksia ei kuitenkaan pyritä arvioimaan määrällisesti koko yhteiskunnallisessa mittakaavassa, vaan tarkastelu keskittyy siihen, miten ohjelmistokehittäjät ja organisaatiot voivat tunnistaa, arvioida ja hallita näitä vaikutuksia ohjelmistokehitysprosessien, arkkitehtuurivalintojen ja vaatimustenhallinnan näkökulmasta. Kestävän elinkaariajattelun tulee olla tietoisesti integroituna osaksi ohjelmistokehityksen suunnittelua, määrittelyä, toteutusta ja elinkaaren aikaista ylläpitoa.

1.6 Tutkielman rakenne

Tämä tutkimus on jaettu kuuteen lukuun, joista tässä esitellään lukujen sisällöt.

Kirjallisuuskatsauksen luku 2.1 käsittelee kestävän kehityksen käsitettä, sen ulottuvuuksia ja sääntelytaustaa sekä yritysten kestävää kehitystä ja kestävää projektinhallintaa. Luvussa 2.2 tarkastellaan kestävää ohjelmistokehitystä, ICT vaikutustasoja, vihreää IT:ä, kestävän ohjelmistokehityksen vaiheita, vesiputousmallia, ketterää ohjelmistokehitystä, ohjelmiston suunnittelua, vaatimusten määrittelyä, kestävyyttä ei-toiminnallisena vaatimuksena, Lean-ajattelua, ohjelmiston elinkaarta, asiakasarvoa, Voice of Customer -näkökulmaa, Customer Utility Map -työkalua sekä ohjelmiston toteutusta. Luvussa 2.3 kootaan kirjallisuuskatsauksen pohjalta tutkimuksen teoreettinen viitekehys.

Luvussa 3 kuvataan laadullista tutkimusta ja perustellaan tähän työhön valitut tutkimusmenetelmät. luvussa 3.2 kuvataan haastateltavat ja perustellaan heidän valintansa. Luku 3.3 selvittää aineiston keruumenetelmät ja aineiston tallentamisen. Luvussa 3.4 käsitellään ja perustellaan aineistoanalyysin valinta tutkimusmenetelmäksi.

Luku 4 käsittelee tutkimuksen empiirisiä tuloksia ja analyysiä. Luvussa tarkastellaan, miten kestävyys ymmärretään ohjelmistokehityksessä, miten se näkyy ei-toiminnallisena vaatimuksena ja millä tavoin se liittyy ohjelmiston elinkaaren vaiheisiin. Lisäksi luvussa analysoidaan ohjelmistojen suoria, epäsuoria ja systeemisiä vaikutuksia, kädenjälkeä, rebound-vaikutuksia, mittareita, arviointikäytäntöjä, organisaation vastuita sekä kestävyyden integrointia vaikeuttavia esteitä.

Luvussa 5 esitetään tutkimuksen teoreettisen viitekehyksen ja empiiristen havaintojen pohjalta muodostettu kestävyysmalli ohjelmistokehitykseen. Mallin tarkoituksena on tukea ohjelmistokehitystiimejä, projektipäälliköitä, arkkitehtejä, tuoteomistajia, asiakkaita ja organisaatioiden vastuullisuustiimejä ohjelmistojen kestävyyteen liittyvien vaikutusten tunnistamisessa, arvioinnissa ja hallinnassa.

Luvussa 6 esitetään tutkimuksen johtopäätökset, arvioidaan tutkimuksen keskeiset tulokset suhteessa tutkimuskysymyksiin sekä kuvataan jatkotutkimusaiheet.

1.7 Generatiivisen tekoälyn hyödyntäminen tutkielmassa

Tätä tutkielmaa tehtäessä on hyödynnetty generatiivista tekoälyä (GPT5.5 Thinking) tutkimus- ja kirjoitusprosessissa aputyökaluna. Tätä tutkimista toteutettaessa generatiivisella

tekoälyllä on annettu työkalun rooli: auttaa haastavissa tilanteissa, tukea ideointia epäselvissä kohdissa ja selkeyttää akateemista ilmaisua. Tutkimuksen alussa tutkimusongelma on ideoitu omien ajatusten, ideoiden ja kandidaatin tutkinnosta nousseiden teemojen pohjalta. Tekoäly auttoi jossakin vaiheessa rönsyävän aiheen rajaamista, tiivistämistä ja tutkimuskysymysten tarkentamista.

Tekoälyä käytettiin tieteellisten tekstien ja aineiston kääntämiseen suomen kielelle. Havaittavaa oli, että kokonaisen laajan tutkimuksen tulkitseminen ei onnistu, vaan teksti täytyy syöttää osissa, jotta lopullinen teksti vastaa englannin kielistä versiota. Tekoäly voi jättää pois keskeisiä osia tutkimuksista tai keksiä sinne omia osioitaan syötettäessä liian laajoja aineistoja. Koska tekoälyllä on tapana jossakin kohtaa lähteä omille teilleen, on tämä keskeinen havainto, jota työssä ei ole voinut sivuuttaa. Syötteen täytyy olla niin yksiselitteinen, että sen tulkinnassa ei ole epäselvyyttä sen antajalla ja tekoäly pystyy vastaamaan siihen yksiselitteisesti.

Tutkijan itse tuottaman tekstin kielellisen selkeyttämiseen on käytetty tekoälyä akateemisen ilmaisun tavoittamiseksi. Kielellisen tekstin parantamiseen työssä usein käytetty tekstin muokkaus käsky kuuluu: " Anna kolme vaihtoehtoa (tekstin pätkä tai lause)”. Tekoälyn tuottamien vastausten arviointi on perustunut teorian ja siihen pohjautuvan koostetun tiedon ja käsitteistön tunnistamiseen. Toisena usein käytettynä syötteenä käytettiin: ”mitä muuttaisit tässä (tekstiä), anna kursiivilla muutokset, anna yliviivattuna poistettavat).

Tekoäly auttoi myös haastattelun ja siihen liittyvän kyselylomakkeen kysymysten hiomiseen sekä tietosuojalausekkeen muodostamiseen haastattelun alussa, haastatteluaineiston koodausrungon rakentamisessa ja löydösten perusteella uusien koodien muodostamisessa.

Haastatteluaineisto analysoitiin vaiheittain välittömästi jokaisen haastattelun jälkeen. Microsoft Teamsilla tallennetut haastattelut litteroitiin transcription.utu.fi -ohjelmalla ja litteroitu teksti muunnettiin selkokieliseksi tekoälyn avulla heti haastattelun jälkeen. Tässä havaintona oli se, että jokaisen haastattelun aineisto täytyy syöttää kahden kysymyksen jaksoissa, jotta tutkija pysyi selkokielelle muokatun haastattelun vaiheissa mukana ja pystyi varmistamaan muodostuvan tekstin oikeellisuuden. Tutkimustyön viimeistelyssä tekoälyä käytettiin viitteiden ja lähdeluettelon korjaamiseen.

2 Kirjallisuuskatsaus

Tässä luvussa esitetään tutkimuksen teoreettinen tausta ja keskeiset käsitteet. Kirjallisuuskatsauksen tarkoituksena on muodostaa viitekehys, jonka avulla kestävyyttä voidaan tarkastella ohjelmistokehityksen suunnittelun, vaatimustenhallinnan, projektinhallinnan ja ohjelmiston elinkaaren näkökulmista.

Tässä luvussa käsitellään ensin kestävän kehityksen yleisiä lähtökohtia, kestävää projektinhallintaa ja kestävää ohjelmistokehitystä. Tämän jälkeen tarkastellaan ICT:n vaikutustasoja, vihreää IT:tä, ohjelmistokehityksen elinkaarimalleja, ei-toiminnallisia vaatimuksia sekä Lean-ajattelun ja asiakasarvon merkitystä kestävyyden tunnistamisessa. Luvun lopussa nämä näkökulmat kootaan tutkimuksen teoreettiseksi viitekehykseksi, jota hyödynnetään haastatteluaineiston analyysissä ja kestävyysmallin muodostamisessa.

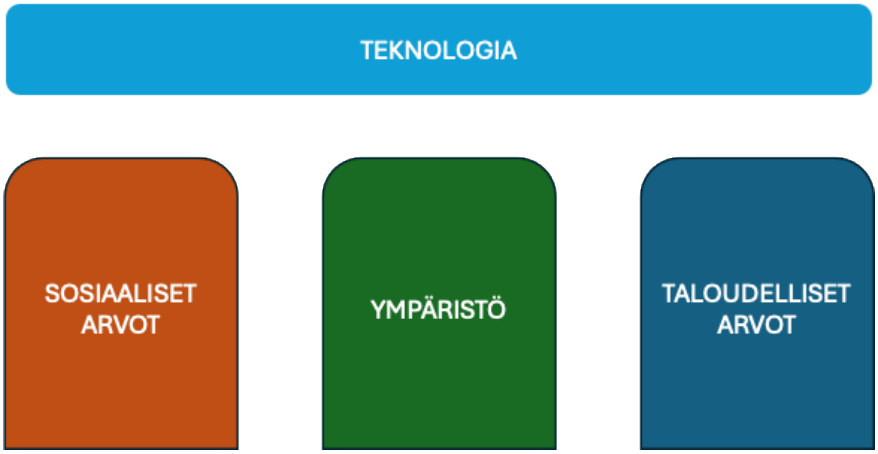
2.1 Kestävä kehitys

Maailman ympäristö- ja kehityskomissio (WCED) määritteli vuonna 1987 kestävän kehityksen seuraavasti: ”kehitys, joka tyydyttää nykyhetken tarpeet vaarantamatta tulevien sukupolvien mahdollisuuksia tyydyttää omat tarpeensa” (WCED, 1987). Toisena keskeisenä kestävän kehityksen käsitteenä on niin kutsuttu triple bottom line -ajattelu, joka esiteltiin 1990-luvun lopulla (Elkington, 1997). Triple bottom line -ajatuksena on jakaa kestävyys kolmeen toisiinsa kytkeytyvään ulottuvuuteen: sosiaaliseen, ympäristölliseen ja taloudelliseen kestävyyteen.

Ahi ja Searcy (2014) korostavat, että kestävyys on luonteeltaan monimutkainen ja moniulotteinen ilmiö, joka ulottuu ympäristöllisiin, taloudellisiin ja sosiaalisiin näkökulmiin. Näitä ulottuvuuksia kuvataan kirjallisuudessa usein kestävän kehityksen pilareina tai periaatteina, ja niiden tarkastelu tarjoaa hyödyllisen viitekehyksen kestävyyden jäsentämiseen ja käytäntöön viemiseen. Keskeinen haaste liittyy kuitenkin siihen, miten näiden pilarien välinen vuorovaikutus ja siitä syntyvä dynaaminen kokonaisuus voidaan ymmärtää ja hallita, erityisesti tilanteissa, joissa yhden ulottuvuuden ratkaisut voivat vaikuttaa toisiin odottamattomilla tavoilla (Penzenstadler ym., 2018). Tämä korostaa kestävyyden laajempia systeemisiä vaikutuksia, minkä vuoksi yksittäisiä toimenpiteitä ei tule arvioida irrallaan niiden mahdollisista vaikutuksista muihin kestävyyden osa-alueisiin.

Kuvio 1. Kestävän kehityksen kolmen ulottuvuuden malli Ahi ja Searcyä (2014) mukaillen.

Kuvio 1 havainnollistaa kestävän kehityksen kolmen pilarin mallia, jossa ympäristö, yhteiskunta ja talous muodostavat toisiinsa kytkeytyvän kokonaisuuden. Malli korostaa, että yksittäisen ulottuvuuden muutoksilla on väistämättä vaikutuksia myös muihin kestävän kehityksen osa-alueisiin, mikä tekee kestävyyden tarkastelusta luonteeltaan systeemistä.



Kuvio 2. Teknologian rooli kestävän kehityksen ulottuvuuksien tukena Moragaa ym. (2019) mukaillen.

Kuvio 2 havainnollistaa teknologian keskeistä roolia kestävän kehityksen muiden osa-alueiden tukena Moragan ym. (2019) mallissa. Teknologia toimii tässä eräänlaisena kattavana sateenvarjona, jonka kautta voidaan toteuttaa ja skaalata yhteiskunnalliset, taloudelliset ja ympäristölliset kehitysaskeleet. Digitaaliset ratkaisut mahdollistavat näiden ulottuvuuksien välisen vuorovaikutuksen ja tarjoavat keinoja kestävyyshaasteiden hallintaan yhä

monimutkaisemmissa toimintaympäristöissä. Ohjelmistoteknologialla on merkittävä rooli tällaisten keskinäisriippuvaisten, monimutkaisten, ja globaalisti hajautettujen järjestelmien suunnittelussa ja hallinnassa, ja sitä voidaan hyödyntää kestävän kehityksen edistämiseen laaja-alaisesti (Penzenstadler ym., 2018). Tämä tarkoittaa, että teknologia voidaan nähdä sekä kestävyyshaasteiden aiheuttajana että niiden ratkaisemisen mahdollistajana.

Yhdistyneiden kansakuntien kestävän kehityksen tavoitteet (Sustainable Development Goals, SDGs) muodostavat globaalin viitekehyksen kestävän kehityksen edistämiseksi. Agenda 2030 -toimintaohjelma sisältää 17 tavoitetta, joiden tarkoituksena on edistää taloudellista kehitystä, vähentää eriarvoisuutta sekä suojella ympäristöä (United Nations, 2015). Kestävän kehityksen tavoitteet nojaavat kolmeen peruspilariin: ympäristölliseen, sosiaaliseen ja taloudelliseen kestävyyteen, ja ne tarjoavat organisaatioille ja yrityksille yleisen viitekehyksen kestävyyden systemaattiseen edistämiseen. Useat kirjallisuuskatsaukset osoittavat, että yritysstrategiaan integroidut kestävän kehityksen tavoitteet korreloivat positiivisesti yrityksen kokonaissuorituskyvyn kanssa. Tästä huolimatta kestävän kehityksen tavoitteiden käytännön jalkauttaminen ja integrointi organisaatioiden toimintaan on edelleen puutteellista (Hristov, 2022). Tämän vuoksi kestävyyden edistäminen edellyttää sekä strategisen tason sitoutumista että käytännön toimintamalleja, jotka mahdollistavat tavoitteiden kääntämisen mitattaviksi ja seurattaviksi toimenpiteiksi (Silvius & Schipper, 2014). Kestävyyden raportointi on tullut vakiintuneeksi käytännöksi erillisenä raporttina tai osana organisaation vuosikertomusta. Tämä edellyttää toiminnan systemaattisten käytänteiden käyttöönottoa kestävyyden huomioimiseksi ja seurannan kehittämiseksi. Raportointiin lisää painetta kansainväliset säädökset ja lainsäädäntö.

Digitaalisilla teknologioilla ja ohjelmistoilla on keskeinen rooli näiden tavoitteiden saavuttamisessa. Ohjelmistot voivat tukea esimerkiksi energiatehokkuuden parantamista, resurssien optimointia, toimitusketjujen läpinäkyvyyttä sekä päätöksenteon tietopohjaa eri toimialoilla (Naumann ym., 2011). Samanaikaisesti ohjelmistojärjestelmät voivat myös lisätä energiankulutusta ja resurssien käyttöä, minkä vuoksi niiden vaikutuksia tulee tarkastella koko elinkaaren näkökulmasta. ICT-ratkaisujen roolia kestävän kehityksen tavoitteiden saavuttamisessa on korostettu useissa tutkimuksissa. Teknologiat voivat toimia mahdollistajina kestävyyssiirtymässä, mutta niiden suunnittelussa ja käytössä tulee huomioida sekä suorat että epäsuorat vaikutukset yhteiskuntaan, ympäristöön ja talouteen. (Hilty & Aebischer, 2015.)

Vuonna 2014 voimaan tullut EU:n ei-taloudellista raportointia koskeva direktiivi (NFRD) velvoittaa yleisen edun kannalta merkittävät yhteisöt laatimaan ei-taloudellisen selvityksen vuosittain. Se sisältää riittävät tiedot yrityksen kehityksen, suorituskyvyn ja taloudellisen aseman arvioimiseksi. Tämän lisäksi siinä arvioidaan toiminnan vaikutuksia ympäristö - ja sosiaaliseen kestävyyteen, kuten henkilöstö, ihmisoikeudet, korruptio ja lahjonnan torjunta. Tämä raportointi koskee yli 500 henkilön organisaatioita (European Parliament and Council of the European Union, 2014).

Tammikuussa 2023 voimaan tullut Yritysten kestävyysraportointidirektiivi (Corporate Sustainability Reporting Directive, CSRD) laajensi suurten organisaatioiden pakollista raportointia. Direktiivi laajensi sosiaalisten ja ympäristöllisten vaikutusten raportointivaatimuksia velvoittamalla yritykset raportoimaan suorat (Scope 1) ja epäsuorat (Scope 2 ja Scope 3) päästöt. Ohjelmistoihin liittyvät päästöt kuuluvat Scope III -luokkaan. (Sipilä ym., 2023.) Kestävyysraportointi on nyt vakiintunut käytäntö ja se toteutetaan joko osana vuosikertomusta tai erillisenä raporttina, joka edellyttää kestävien toimintatapojen ja kestävyyden seuraamista projekteissa ja muussa toiminnassa.

Ympäristövastuulla viitataan yksilön tai organisaation eettiseen ajattelutapaan ja periaatteisiin, joka korostaa ympäristöön kohdistuvan haitan minimoimista ja välttämistä, sekä vastuun ottamista omasta toiminnasta ja sen seurauksista. Organisaatiossa tehtäviä päätöksiä ja toimenpiteitä ohjaavat ensisijaisesti yksilöiden asenteet ja ajattelutavat, eivätkä pelkästään sääntelyyn ja asiakasvaatimuksiin reagoiminen. Eettinen ajattelutapa ja periaatteet tarkoittavat, että ympäristöön kohdennetut ponnistelut tehdään harkitusti ja tietoisesti. (Murugesan, 2008; Lozano, 2012b.)

Yritysvastuudirektiivi (Corporate Sustainability Due Diligence Directive, CSDD) asettaa yrityksille velvoitteen tulevaisuudessa tunnistaa ja ehkäistä kielteisiä ympäristövaikutuksia sekä ihmisoikeusloukkauksia. Viherväittämiä koskevan direktiivi, (Directive on Empowering Consumers for the Green Transition) tulee sovellettavaksi asteittain vuodesta 2026 alkaen EU:ssa ja sen tavoitteena on ehkäistä viherpesua. Tämä direktiivi tuo sitovia vaatimuksia vääristä tai harhaanjohtavista ympäristöväitteistä markkinoinnissa. Sääntelykehitys lisää painetta kehittää myös käytännön toimintamalleja ja mittaamista, joiden avulla vaikutuksia voidaan tunnistaa ja raportoida johdonmukaisesti.

2.1.1 Yritysten kestävä kehitys

Organisaatiot tarvitsevat hallintajärjestelmiä, hyviä käytäntöjä sekä yhteistyötä muiden organisaatioiden kanssa varmistaakseen, että niiden toiminta, projektit ja päätöksenteko tukevat kestävyyden tavoitteita. Yritysten kestävä kehitys (CorporateSustainability, CS) määritellään seuraavasti: ”Yritystoiminnot, joilla pyritään ennakoivasti edistämään kestävän kehityksen tasapainoa, mukaan luettuina nykypäivän taloudelliset, ympäristölliset ja sosiaaliset ulottuvuudet sekä niiden keskinäiset suhteet aikaulottuvuuden sisällä ja sen aikana (eli lyhyellä, pitkällä ja pidemmällä aikavälillä), ja jotka kohdistuvat yrityksen järjestelmiin eli toimintaan ja tuotantoon, johtamiseen ja strategiaan, organisaatiojärjestelmiin, hankintoihin ja markkinointiin sekä arviointiin ja viestintään sekä sidosryhmiin”. (Lozano, 2011.)

Yritykset ja niiden johtajat ovat yhä tietoisempia taloudellisten, ympäristöön liittyvien ja sosiaalisten näkökohtien välisistä suhteista ja keskinäisistä riippuvuuksista (CEC, 2001; Elkington, 2002) sekä toimintansa lyhyen, pitkän ja pidemmän aikavälin vaikutuksista (Lozano, 2008b) eli kestävyyden neljästä ulottuvuudesta (taloudellinen, ympäristöllinen, sosiaalinen ja ajallinen ulottuvuus) ja niiden vuorovaikutuksesta. Kestävän kehityksen periaatteiden sisällyttäminen yrityksen järjestelmään on kuitenkin merkittävä haaste, erityisesti niiden monimutkaisuuden ja moniulotteisuuden vuoksi (Langer & Schön, 2003).

Lisäksi useissa lähestymistavoissa korostuvat teknologiakeskeiset ratkaisut ja hallinnolliset mekanismit, minkä seurauksena yrityskulttuuri, toimitusketju sekä yritysjärjestelmän elementtien ja kestävyyden neljän ulottuvuuden väliset suhteet jäävät usein tarkastelun ulkopuolelle (Lozano, 2012b). Siebenhunerin ja Arnoldin (2007) mukaan, jotta yrityksestä tulisi kestävyyssuuntautuneempi, sen olisi tehtävä muutoksia, joihin kuuluu resurssitehokkaiden teknologioiden käyttöönotto, kestävyysraportointijärjestelmät sekä kestävien tuotteiden, palvelujen ja tuote-palveluyhdistelmien tarjoaminen. Systeemien tutkiminen tai systeemiajattelu voi auttaa ymmärtämään organisaation ja organisaatioiden keskinäisiä riippuvuuksia, vuorovaikutusta ja kytkeytyneisyyttä toisiinsa, organisaation osien välisten ja organisaatioiden välisten rajojen merkitystä sekä yksilöiden rooleja rajojen sisällä ja niiden yli (Stacey, 1993).

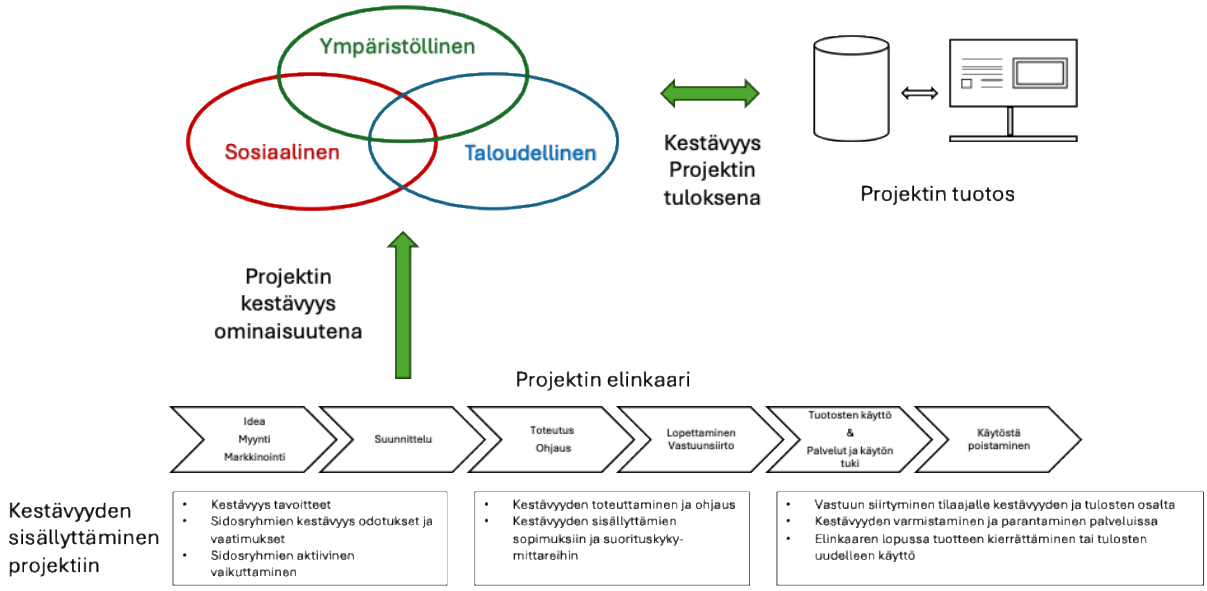
2.1.2 Kestävä projektinhallinta

Toimitusprojektien kestävyyttä voidaan tarkastella useasta näkökulmasta. Tutkimuksessa on tunnistettu kestävyyden neljä osa-aluetta: ihmiseen liittyvät, organisaatioon liittyvät, prosessiin liittyvät ja tuotokseen liittyvät näkökulmat (Marcelino-Sádaba ym., 2015). Projektiliiketoiminnan kestävyys tarkoittaa kestävyyttä toteutusprosessissa sekä projektin tuotoksessa (Gareis ym., 2013). Tämä tarkoittaa, että projektin kestävyyttä arvioidaan sekä sen lopputuloksen että projektin suunnittelu- ja toteutusprosessien näkökulmasta.

Projektin tuotokseen vaikuttaa myös kestävä projektin johtaminen, koska projekti suunnitellaan ja toteutetaan projektitoimituksen aikana. Kestävä projektin hallinta tarkoittaa toteutuksen suunnittelemista, ohjausta ja valvontaa, joka huomio ympäristölliset, sosiaaliset ja taloudelliset tekijät projektin elinkaarella eri sidosryhmien kannalta. Se tarkastelee prosesseja, resursseja, tuotoksia ja niiden vaikutuksia laaja-alaisesti huomioon ottaen sidosryhmien odotukset, osallistamisen ja sen vaikutukset sekä pyrkien tuottamaan hyötyä sidosryhmille läpinäkyvällä, eettisellä ja reilulla tavalla (Silvius & Schipper, 2014, s. 79). Kestävä projektin johtaminen rakentuu ja edellyttää yhteistyötä sidosryhmien kanssa (Eskerod & Huemann, 2013) tasapainottaa kolmea kestävyyden ulottuvuutta (Silvius & Schipper, 2014) sekä sisältää elinkaariajattelun, jossa huomioidaan projektin lisäksi myös sen tuotoksen ja käytön aikaiset vaikutukset (Labuschagne & Brent, 2005). Kestävä projektinhallinta laajentaa perinteistä projektinhallinnan näkökulmaa huomioimaan projektin vaikutukset koko sen elinkaaren aikana. Projektin johtamisesta puuttuu kestävyys useista syistä: kestävyyden matalat taloudelliset hyödyt suhteessa vaadittuun investointiin, keskeisten sidosryhmien sitoutumisen puute, eturistiriidat ja muuttuvat olosuhteet. Kestävää projektin johtamista sen sijaan on esitetty kuvaamaan eri tyylisiä käytänteitä: kestävyys ilmaistaan selkeänä arviointikriteerinä, kokonaisvaltainen projektin suunnittelu sisällyttäen kestävyys lopputulokseen, sidosryhmien odotusten ja huolien tarkastelu ja projektitoimituksen joustavuuden varmistaminen arvon kasvattamiseksi. (Kivilä ym., 2017.)

Projektin ohjausta voidaan lähestyä kokonaisvaltaisesti käyttäytymistieteellisestä näkökulmasta ja nojautua organisaatio-ohjaukseen, joka muodostuu pysyvästä organisaatiosta sekä esihenkilö-alaissuhteesta. Kokonaisvaltainen ohjaus voi ilmetä sekä erilaisina ohjausmuotoina, kuten formaalina ja epäformaalina ohjauksena, että erilaisina ohjausmekanismeina, kuten suunnitelmina, budjetteina, sosiaalisena ohjauksena ja säännöstöinä. Projektin toteutusvaiheessa tavoitteiden saavuttaminen perustuu usein selkeiden

suorituskykymittareiden määrittelyyn sekä diagnostisten projektinohjaustyökalujen käyttöön, kuten arvon analyysiin ja projektin terveystarkastuksiin. (Kivilä ym., 2017.)



Kuvio 3. Kestävyyden sisällyttäminen projektiin, mukaillen Martinsuo & Vuorinen (2024)

Organisaatiot ja yritykset tarvitsevat uusia tapoja projektin johtamiseen, jotta kestävyys saadaan sisällytettyä prosesseihin siirtyäkseen niin sanotun rautaisen kolmion (kustannusten, ajan ja laadun) laajempiin vaikutuksiin (Silvius & Schipper, 2014). Kuviossa 4 ovat rautaisen kolmion projektin aikaiset riippuvuudet kuvattuna. Kestävyys täytyy huomioida erityisesti toimitusprojekteissa, joissa lopputuotteella ja prosessilla on merkittäviä sosiaalisia ja ympäristöllisiä vaikutuksia. Lopputuotoksen kestävyyden arviointi ei riitä, vaan myös toteutusprosessin täytyy olla kestävä.

Kuvio 4. Projektinhallinnan rautakolmio. (GPT5.5)

Kestävän projektijohtamisen käytäntönä tulee olla kannattava, läpinäkyvä, eettinen, oikeudenmukainen ja ympäristöystävällinen projektin toteutus, jolla pyritään sosiaalisesti ja ympäristöllisesti hyväksyttävään lopputulokseen koko lopputuotteen elinkaaren ajan (Silvius & Schipper, 2014).

”Kestävä projektinhallinta (SPM) on projektitoimituksen ja tukiprosessien suunnittelua, seurantaa ja ohjausta siten, että huomioidaan projektin resurssien, prosessien, toimitusten ja vaikutusten elinkaaren ympäristölliset, taloudelliset ja sosiaaliset näkökohdat, tavoitteena tuottaa hyötyjä sidosryhmille, ja toteutettuna läpinäkyvällä, oikeudenmukaisella ja eettisellä tavalla, joka sisältää proaktiivisen sidosryhmien osallistamisen” (Silvius & Schipper, 2014, s. 79). Tätä määritelmää voidaan soveltaa myös ohjelmistoprojekteihin, joissa projektituotos voi muuttaa prosesseja ja käyttäytymistä laajasti, jolloin kestävyyden arviointi ei rajoitu pelkkään projektin toteutuksen hiilijalanjälkeen.

Kestävyyssiirtymän ja siihen liittyvien strategioiden toteuttaminen edellyttää muutoksia henkilöstön ajattelutavoissa siten, että työntekijät sitoutuvat edistämään kestävyyttä omassa työssään ja arjessaan (Vuorinen, 2024). Tämä korostaa niin sanottujen pehmeiden tekijöiden, kuten asenteiden, osaamisen ja sitoutumisen merkitystä kestävyyden integroimisessa projektitoimintaan.

Kuten Gareis ym. (2013) edellä esittävät, projektien kestävyyttä voidaan tarkastella kahdesta toisiaan täydentävästä näkökulmasta: projektin toteutusprosessin kestävyytenä sekä projektin lopputuotoksen kestävyytenä. Yhdessä nämä muodostavat kokonaisvaltaisen viitekehyksen projektin kestävyyden arvioinnille. Tämä jaottelu on relevantti myös ohjelmistoprojektien kontekstissa, koska ohjelmiston tuotokseen liittyvät kestävyyshyödyt tai -haitat realisoituvat usein vasta järjestelmän käyttövaiheessa.

Projektin kestävyys korostaa projektinhallinnassa käytettävien prosessien kestävyyttä, eli sitä, toteutetaanko projekti kestävien toimintatapojen mukaisesti siten, että ympäristölle aiheutuvat haitat ovat mahdollisimman vähäisiä (Sabini ym., 2019). Projektinhallinnan prosessit voivat siten olla kestäviä, vaikka projektin lopputulos ei itsessään olisi kestävä. Tämä erottelu on tärkeä, koska se auttaa välttämään oletuksen, että ”vihreä projekti” johtaisi automaattisesti ”vihreään lopputulokseen”, ja tuo esiin arvioinnin kaksi erillistä tasoa.

Kestävä projektijohtaminen sekä kestävyyden tarkastelu arvonluonnin näkökulmasta nostavat kestävyyden keskeiseksi tavoitteeksi tai vähintään yhdeksi projektin tavoitteiden tärkeäksi

osatekijäksi (Vuorinen, 2024). Tämä korostaa projektin alkuvaiheen merkitystä, jossa määritellään projektin tavoitteet, laajuus ja vaikutukset. Näin ollen kestävyyttä tulisi tarkastella jo projektin front end -vaiheessa esimerkiksi tavoitteiden, rajauksien, hyväksymiskriteerien ja vaikutusten arvioinnin kautta.

Projektin aivan varhaisimpiin vaiheisiin viitataan termillä project front end, jotka edeltävät yksityiskohtaista projektisuunnittelua ja toteutusta (Vuorinen ym., 2024 s.10). Projektin alkuvaiheen yksi keskeisiä tehtäviä tulevassa projektissa on tavoiteltavan lopputuloksen määrittely (Williams ym., 2019). Tämä tarkoittaa käytännössä projektin laajuuden ja tavoitteiden määrittelyä. Projektin elinkaaren myöhemmissä vaiheissa painopiste siirtyy kestävän projektijohtamisen toteutumisen, kestävän projektitoteutuksen ja kestävän lopputuloksen varmistamiseen. Käyttövaiheen kestävyyttä voidaan tarkastella projektituotosten käytön kestävyyden ja sen odotetun elinkaaren näkökulmasta.

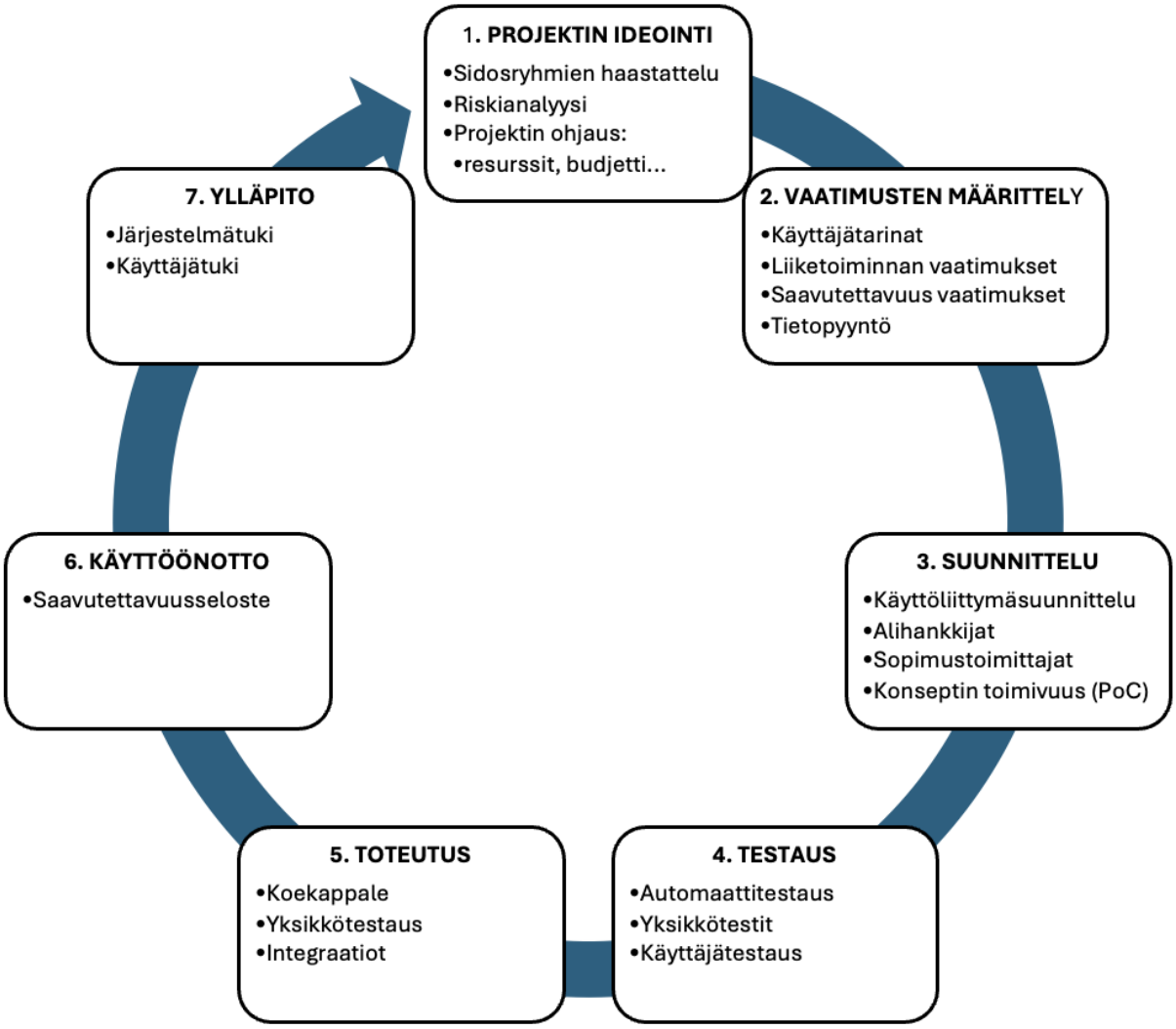
Tietojärjestelmähankkeiden onnistumista koskevassa selvityksessä hankinnan kunnollinen resursointi ja valmistelu sekä tilaajan ja toimittajan välinen viestintä nousivat keskeisiksi onnistumistekijöiksi (Celkee Oy, Tietotekniikan liitto ry, & Ohjelmistoyrittäjät ry, 2013). Tämä tukee kestävän projektinhallinnan näkökulmaa, jossa projektin alkuvaiheen määrittely, sidosryhmien osallistaminen, resurssien varmistaminen ja vastuunjaon selkeys vaikuttavat olennaisesti projektin lopputulokseen. Ohjelmistoprojekteissa tämä on tärkeää myös kestävyyden kannalta, koska kestävyyteen liittyvät tavoitteet, hyväksymiskriteerit ja vastuut on määriteltävä jo ennen varsinaista toteutusta.

Kokonaisvaltainen lähestymistapa kestävyyttä tukevaan toimintaan merkitsee osaamisen, asenteiden ja käyttäytymisen muutosta niin, että kaikki sitoutuvat kestävyyttä edistävään toimintaan työssään ja arjessaan (Vuorinen 2024). Ohjelmistoprojekteissa tämä korostuu erityisesti, koska järjestelmien keskeiset ympäristö- ja yhteiskunnalliset vaikutukset realisoituvat usein vasta järjestelmän käyttövaiheessa.

2.2 Kestävä ohjelmistokehitys

Keskeinen rooli kestävyydessä on vaatimusten määrittelijöillä. Heitä voidaan sanoa kestävyysinsinööreiksi, jotka ylittävät kapean järjestelmänäkökulman ja omaksuvat sidosryhmäkeskeisen, systeemiorientoituneen ja monitieteellisen lähestymistavan ja heillä on ylimmän yritysjohdon tuki. Heidän tehtävänään on ymmärtää ohjelmistointensiiviset järjestelmät osana laajempia kokonaisuuksia ja arvioida niiden vaikutuksia sosiaalisiin,

teknisiin, taloudellisiin ja luonnollisiin järjestelmiin sekä yksilöihin näissä järjestelmissä. (Becker ym., 2015).



Kuvio 5. Ohjelmistokehityksen elinkaarimalli University of Melbournea (2011) mukaillen.

Kuvio 5 havainnollistaa, että nykyiset ohjelmistokehityksen elinkaarimallit eivät sisällä kestävyyden näkökulmaa, mikä on rajoittanut organisaatioiden ja ohjelmistokehittäjien mahdollisuuksia integroida kestävyys osaksi kehitysprosessia kestävien järjestelmien tuottamiseksi sekä niiden ympäristövaikutusten arvioimiseksi. Ohjelmistojen vaikutukset ulottuvat kuitenkin teknologian käytön seurauksena laajasti eri yhteiskunnallisiin ja ympäristöllisiin järjestelmiin, minkä vuoksi niiden tarkastelu edellyttää laajempaa systeemistä näkökulmaa.

Kuvio 6. Kestävän ohjelmiston keskeiset näkökulmat Abdullateef Shola (2016) mukaillen.

Moraga ym. (2019) laajentavat kestävyyden lähestymistapaa jakamalla sen neljään eri ulottuvuuteen: ympäristölliseen, taloudelliseen, sosiaaliseen ja tekniseen. Heidän mukaansa teknologialla on keskeinen rooli toimia välittävänä elementtinä, jonka kautta palveluiden, materiaalien, ja tuotteiden elinkaaren syklit vaikuttavat yhteiskuntaan, ympäristöön ja talouteen. Penzenstadler ym. (2018) ja Karlskronan manifestissään Becker ym. (2015) lisäävät kestävyyden kolmeen pilariin kaksi ulottuvuutta: teknisen, jolla viitataan ohjelmistojen elinkelpoisuuteen pitkällä aikavälillä ja yksilöllisen, joka merkitsee henkilökohtaista autonomiaa ja hyvinvointia. Näin kestävyyden tarkastelu laajenee perinteisestä kolmen pilarin mallista kohti moniulotteista lähestymistapaa, jossa tekniset ratkaisut, yhteiskunnalliset vaikutukset ja yksilölliset kokemukset kytkeytyvät toisiinsa. Olennaista on tunnistaa ja ymmärtää näiden ulottuvuuksien väliset suhteet ja niiden keskinäinen riippuvuus, sillä ne muodostavat yhtenäisen kokonaisuuden. Karlskronan manifestissä Becker ym. (2015) korostaa että kestävyys on monitieteellistä ja kestävyyttä tavoitteleva ohjelmisto suunnittelu edellyttää ohjelmistotekniikassa muiden tieteenalojen käsitteiden ymmärtämistä ja poikkitieteellistä yhteistyötä.

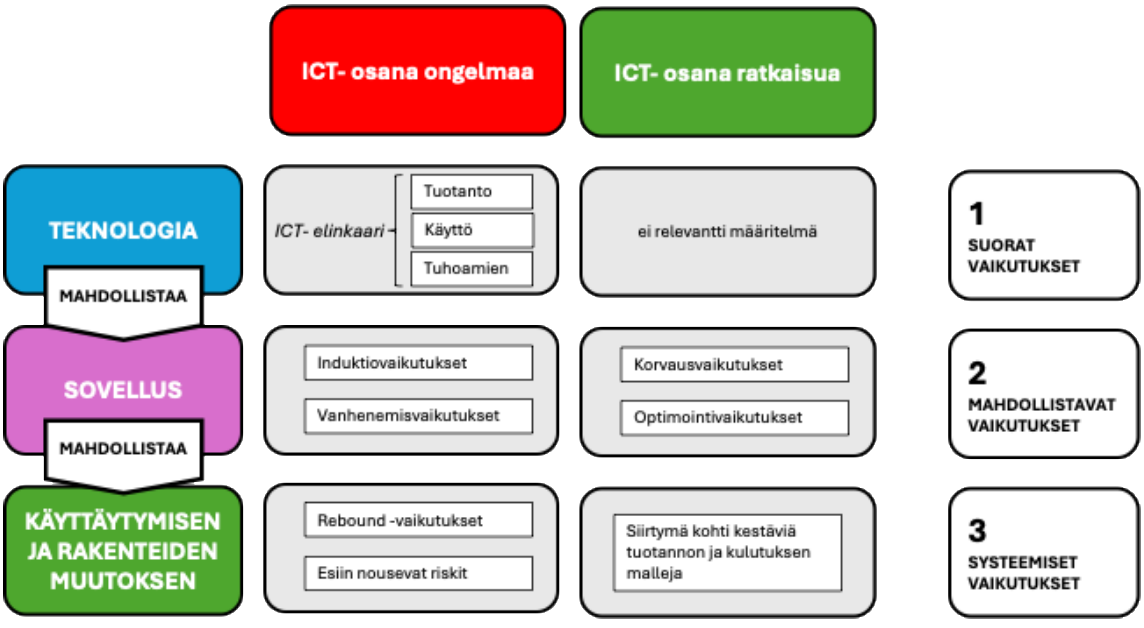
Kestävyys nähdään monitasoisena ilmiönä, mikä edellyttää, että järjestelmän suunnittelua tarkastellaan samanaikaisesti sekä suunniteltavan järjestelmän ja sen kestävyyden näkökulmasta, että sen laajemman järjestelmän näkökulmasta, jonka osaksi se sijoittuu. Sustainability Awareness Framework (SusAF) tarjoaa viitekehyksen ohjelmistojen kestävyyden tarkasteluun ja käsittelee edellä mainittua viittä ulottuvuutta kysymysten muodossa (Duboc ym. 2020). SusAF-viitekehys auttaa tunnistamaan kestävyyteen liittyviä

vaikutuksia järjestelmä- ja systeemitasolla sekä tukee kestävyyden huomioimista ohjelmistosuunnittelun alkuvaiheessa.

2.2.1 ICT:n vaikutustasot

Kestävyysprojektin kautta korostaa projekteissa syntyviä tai niiden mukana tuotettuja kestävyysvaikutuksia, toisin sanoen projekti tuottaa kestävän toimituksen (Sabini ym., 2019). Havainnollistavana esimerkkinä tästä voidaan pitää lentoliikenteen reittien uudelleenjärjestelyä tekoälymallien avulla, jonka mallinnus vähensi ilmastolle haitallisten sumupilvien muodostumista (Teoh ym., 2022). Sumupilvet muodostuvat lentokoneen lentäessä kosteuskerroksen läpi ja nämä sumupilvet sitovat auringonvaloa itseensä, joka poistuisi normaaliolosuhteissa pois ilmakehästä, aiheuttaen ilmaston lämpenemistä.

Hilty ja Aebischer (2015) ovat luokitelleet tieto- ja viestintäteknologian vaikutukset kolmeen eri tasoon: ensimmäisen, toisen ja kolmannen asteen vaikutuksiin. Ensimmäisen asteen vaikutukset liittyvät ICT-laitteiden valmistukseen, käyttöön ja hävittämiseen. Toisen asteen vaikutukset syntyvät ICT:n käytön mahdollistamista toimintatapojen muutoksista, kuten prosessien digitalisoitumisesta ja resurssitehokkuuden parantumisesta. Kolmannen asteen vaikutukset puolestaan liittyvät laajempiin yhteiskunnallisiin muutoksiin, joita ICT:n käyttöönotto voi pitkällä aikavälillä aiheuttaa. Näitä vaikutustasoja on myöhemmin sovellettu myös ohjelmistojärjestelmien kestävyyden tarkasteluun (esim. Naumann ym., 2011), mutta tässä yhteydessä niitä tarkastellaan yleisenä ICT-vaikutusten luokitteluna.



Kuvio 7. ICT: n kolmen vaikutustason matriisi Hiltyn & Aebischerin (2015) mukaan.

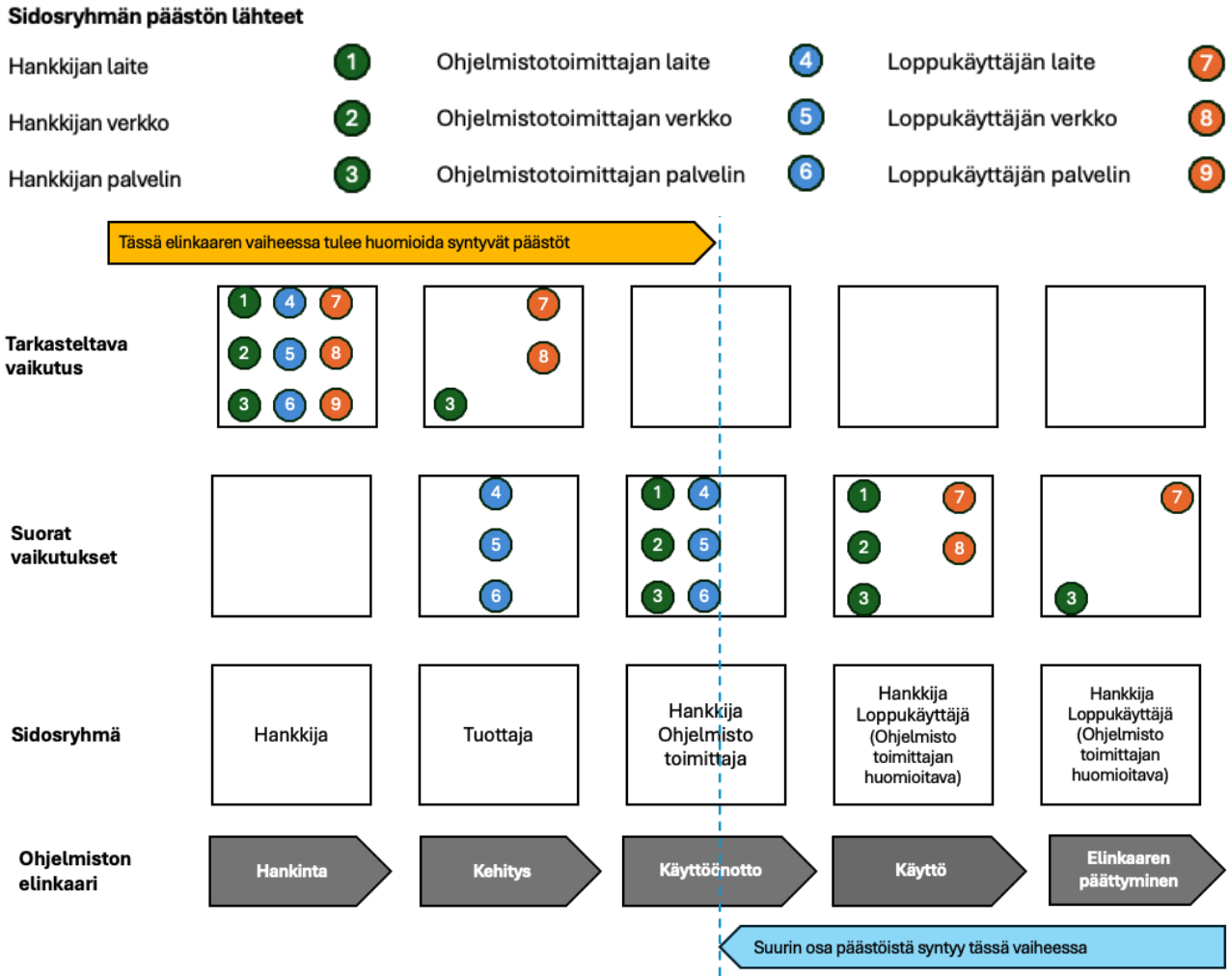
ICT:n kolme vaikutustasoa Hiltyn & Aebischerin (2015) mukaan:

- Ensimmäisen asteen (primaarivaikutukset) viittaavat ICT:n fyysisen olemassaolon aiheuttamiin suoriin ympäristövaikutuksiin. Näihin kuuluvat alustaratkaisun toiminnan edellyttämien datakeskusten energiankulutus ja päästöt, kuten palvelinten käyttö, tietoverkkojen ylläpito sekä jäähdytysjärjestelmien toiminta, samoin kuin ICT-laitteiston tuotannon, käytön, kierrätyksen ja käytöstä poiston ympäristövaikutukset.

- Toisen asteen (sekundaarivaikutukset) kuvaavat ICT:n välillisiä vaikutuksia, jotka syntyvät sen kyvystä muuttaa ja tehostaa prosesseja. Alustatalouden kontekstissa järjestelmä voi esimerkiksi mahdollistaa kulutuskysynnän kasvun ja logistiikkaprosessien nopeutumisen, mikä puolestaan lisää tuotteiden valmistuksesta ja kuljetuksista aiheutuvia päästöjä. Näiden vaikutusten suunta voi olla joko myönteinen tai kielteinen riippuen siitä, miten prosessimuutokset toteutetaan.

- Kolmannen asteen (tertiääri vaikutukset) ilmenevät pidemmällä aikavälillä käyttäytymisen ja taloudellisten rakenteiden sopeutuessa ICT:n ja sen tarjoamien palvelujen pysyvään saatavuuteen. Nopeiden toimituspalvelujen helppous voi muuttaa kuluttajakäyttäytymistä ja yhteiskunnallisia odotuksia siten, että nopeus korostuu arvona, mikä voi johtaa infrastruktuurin laajenemiseen ja resurssi-intensiivisemmän logistiikan vakiintumiseen sekä kasvaviin ympäristövaikutuksiin.

Hilty ja Aebischer (2015) sisällyttävät mallissaan tarkasteluun myös ohjelmistojen hiilikädenjälkivaikutukset eli ne hyödyt, joita ohjelmistot tuottavat eri toimialoille. Lisäksi he korostavat, että systeemisiä ilmiöitä analysoitaessa on olennaista huomioida energiankulutukseen ja CO_2-päästöihin liittyvissä ohjelmistokysymyksissä myös ilmiön vastakkainen puoli.



Kuvio 8. SRoSE-viitekehys ohjelmistopäästöjen jaetun vastuun tarkasteluun.

Partanen ym. (2025) esittelevät ensimmäisinä Shared Responsibility of Software Emissions (SRoSE) -viitekehystä, jossa tarkastellaan ohjelmistoihin liittyvien päästöjen jakautunutta vastuuta (kuvio 8). Viitekehys rakentuu kolmen sidosryhmän, kolmen päästölähteen ja viiden elinkaarivaiheen ympärille. Sidosryhmiksi määritellään hankkija, ohjelmistontoimittaja ja loppukäyttäjä, ja päästölähteiksi tunnistetaan laitteet, verkot ja palvelimet.

Ohjelmiston elinkaari puolestaan koostuu hankinnasta, kehityksestä, käyttöönotosta, käytöstä ja elinkaaren päättymisestä. SRoSE-viitekehys tunnistaa kunkin elinkaarivaiheen ja päästölähteen osalta huomioitavat suorat vaikutukset sekä esittää vastuullisen sidosryhmän kussakin vaiheessa (Partanen ym. 2023).

Partanen ym. (2025) tuovat artikkelissaan useasti esille, että ICT:n vaikutusten ulottuvuuksista ja vaikutusmekanismeista tulee olla tietoinen. He ehdottavat muistisäännöksi **2–3–5**, jossa luku kaksi kertoo vipuvaikutusten kahdesta puolesta: negatiivisesta

hiilijalanjäljestä ja positiivisesta hiilikädenjäljestä. Luku kolme viittaa kolmen tasoisiin vaikutuksiin: suoriin, epäsuoriin ja systeemisiin vaikutuksiin (Hilty & Aebischer, 2015). Luku viisi tarkoittaa kestävyyden viittä ulottuvuutta: ympäristöllinen, sosiaalinen, taloudellinen, tekninen ja yksilöllinen (Penzenstadler ym.,2014). Muistisääntö yhdistää keskeiset kestävän ohjelmistokehityksen analyysikehykset yhdeksi kokonaisuudeksi.

Penzenstadler (2013) määrittelee kestävyyden käsitteen ”kyvyksi kestää”. Edellä esitetyn kuvion 8 perusteella kestävää ohjelmistoa voidaan tarkastella kolmesta eri näkökulmasta. Pitkäikäinen ohjelmisto viittaa siihen, missä määrin ohjelmisto kykenee sopeutumaan ja säilyttämään toimintakykynsä muutostilanteissa. Lean-ohjelmisto puolestaan tarkoittaa ratkaisua, joka edellyttää vähemmän laitteistoresursseja ja pienentää omaa energiankulutustaan, eli on energiatehokas.

Green by IT viittaa näkökulmaan, jossa tarkastellaan IT:n mahdollisuuksia pienentää muiden toimialojen ympäristöjalanjälkeä (Naumann ym., 2011). Lisäksi Digital Sustainability ja ICT for Sustainability (ICT4S) laajentavat tarkastelun ympäristönäkökulmaa pidemmälle ja sisällyttävät kestävyyden kokonaisuuteen myös sosiaaliset näkökohdat (Hilty & Aebischer, 2015). Tämä osoittaa ohjelmiston merkityksen kestävyyden kannalta ja sen valtavan vaikutuksen maailmassa.

GREENSOFT-mallin tavoitteena on tukea ohjelmistokehittäjiä, ylläpitäjiä ja käyttäjiä ohjelmistojen luomisessa, ylläpidossa ja käytössä kestävämmällä tavalla. Yksikään näistä malleista ei kuitenkaan ole tarjonnut ratkaisua kestävyyden lisäämiseksi laatuattribuuttina ohjelmistojärjestelmiin. Tästä syystä on tärkeää tutkia, kuinka kestävyys voidaan kvantifioida laatuattribuuttina ja kuinka kestävyyden tehokkuutta laatuattribuuttina ohjelmistojärjestelmissä voidaan arvioida mittareiden avulla, jotka tuottavat konkreettisia tuloksia siitä, kuinka hyvin kehitysprosessi tuottaa kestäviä ohjelmistoja (Naumann ym., 2014).

Ohjelmiston koko elinkaaren kattava kehdosta hautaan -ajattelu on keskeinen osa GREENSOFT-mallia, joka jäsentää ohjelmiston kehittämiseen liittyvät käsitteet, prosessit ja toiminnot varhaisista määrittely- ja suunnitteluvaiheista aina järjestelmän käytöstä poistamiseen saakka. Mallissa ohjelmistokehityksen kestävyyteen liittyvät vaikutukset jaotellaan kolmeen tasoon: ensimmäisen asteen vaikutuksiin, jotka kuvaavat ICT-ratkaisun toteutuksesta aiheutuvia suoria vaikutuksia, toisen asteen vaikutuksiin, jotka liittyvät ICT:n käytön aikaisiin ja mahdollistaviin vaikutuksiin, sekä kolmannen asteen vaikutuksiin, jotka

ilmenevät laajempina systeemisinä ja kerrannaisina muutoksina yhteiskunnassa ja toimintaympäristössä (Naumann ym., 2014).

2.2.2 Vihreä IT

Green IT määritellään tietokoneiden, palvelimien ja niihin liittyvien alijärjestelmien suunnittelun, valmistuksen, käytön ja käytöstä poiston tutkimukseksi ja käytännöksi siten, että toiminta toteutetaan tehokkaasti, vaikuttavasti ja ympäristövaikutukset pidetään mahdollisimman vähäisinä tai poistetaan kokonaan (Murugesan, 2008). Tämä lähestymistapa korostaa erityisesti energiatehokkuuden parantamista ja laitteiden käyttöasteen optimointia esimerkiksi energiatehokkaiden sirujen suunnittelun, virtualisoinnin, konesalien energiankulutuksen vähentämisen, uusiutuvan energian hyödyntämisen sekä elektroniikkajätteen pienentämisen keinoin (Watson ym., 2008, s. 2).

Vihreäksi ja kestäväksi luonnehditun ohjelmistotuotteen tulisi itsessään olla mahdollisimman kestävä, mikä tarkoittaa, että sen koko elinkaaren aikana aiheutuvien taloudellisten, yhteiskunnallisten, ekologisten sekä ihmisiin kohdistuvien vaikutusten tulisi jäädä mahdollisimman vähäisiksi (Naumann ym., 2011). Kestävyyden näkökulmasta ICT:ssä voidaan erottaa kaksi keskeistä osa-aluetta: Green by IT, joka tarkastelee kestävyyden edistämistä ICT:n avulla, sekä Green IT, joka keskittyy ICT:n oman kestävyyden parantamiseen. Green IT ja Green by IT muodostavat kaksi keskeistä näkökulmaa ICT:n kestävyyden tarkastelussa: ensimmäinen keskittyy ICT:n oman ympäristöjalanjäljen pienentämiseen ja toinen ICT:n mahdollistamiin kestävyyshyötyihin muilla toimialoilla (Murugesan, 2008; Hilty&Aebischer, 2015).

Yhtä olennainen näkökulma koskee energian ja resurssien säästöä, joka voidaan saavuttaa ICT:n soveltamisella muihin tuotteisiin ja palveluihin. Laajemmassa viitekehyksessä tarkastellaan kuitenkin myös sitä, miten ekologiaan, yhteiskuntaan, ihmisiin ja talouteen kohdistuvia kielteisiä vaikutuksia voidaan lieventää ja vastaavasti myönteisiä vaikutuksia edistää (Naumann, 2011).

Kestävyyden tarkasteleminen ohjelmistojärjestelmien laatuattribuuttina ei rajoitu pelkästään energiatehokkuuteen, joka liittyy ohjelmistojen ensimmäisen asteen vaikutuksiin. Ohjelmistokehityksessä kestävyys tulee huomioida ei-toiminnallisena vaatimuksena eli laatuattribuuttina samalla tavoin kuin esimerkiksi tietoturva, käytettävyys ja siirrettävyys, jotta myös toisen ja kolmannen asteen vaikutukset voidaan ottaa huomioon

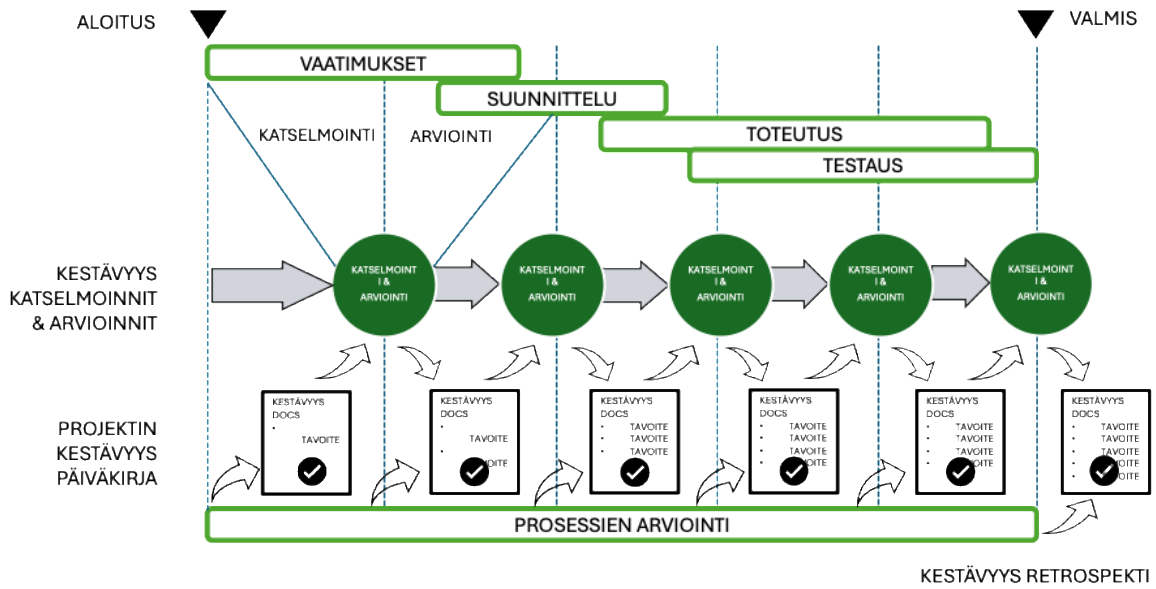
järjestelmäkontekstissa, vaikka niiden arviointi on haastavaa. Tällainen lähestymistapa mahdollistaa ohjelmistojen kestävyyden merkittävän parantamisen taloudellisesta, sosiaalisesta ja ympäristöllisestä näkökulmasta.

Erilaisia vihreän ohjelmiston malleja ohjelmistojärjestelmien kestävyyden parantamiseksi on esitetty runsaasti, erityisesti usein viitattu Naumann ym. (2014) esittämään GREENSOFT -malliin, jonka tavoitteena on tukea ohjelmistokehittäjiä, ylläpitäjiä ja käyttäjiä ohjelmistojen luomisessa, ylläpidossa ja käytössä kestävämmällä tavalla. Yksikään näistä malleista ei kuitenkaan ole tarjonnut ratkaisua kestävyyden lisäämiseksi laatuattribuuttina ohjelmistojärjestelmiin.

Naumann (2011) esittää vihreää ja kestävää ohjelmistoa koskevan määritelmän seuraavasti: ”Vihreä ja kestävä ohjelmisto on ohjelmisto, jonka talouteen, yhteiskuntaan, ihmisiin ja ympäristöön kohdistuvat suorat ja epäsuorat kielteiset vaikutukset, jotka syntyvät ohjelmiston kehittämisestä, käyttöönotosta ja käytöstä, ovat minimaaliset ja/tai jolla on myönteinen vaikutus kestävään kehitykseen”

Vihreän ja kestävän ohjelmistotuotteen toteutuminen edellyttää kuitenkin, että kehittävä organisaatio tunnistaa ne kielteiset ja myönteiset kestävyyteen kohdistuvat vaikutukset, joita ohjelmiston käytöstä todennäköisesti seuraa (Hilty&Aebischer, 2015). Näiden vaikutusten tunnistamiseksi eri sidosryhmien on tarpeen institutionalisoida niiden arviointi ja tunnistaminen osaksi sovellettavia ohjelmistokehitysprosesseja. Tällainen lähestymistapa tekee kestävyyteen liittyvistä kysymyksistä hallittavia ja mahdollistaa sen, että ohjelmistoarkkitehdit, suunnittelijat ja kehittäjät voivat systemaattisesti optimoida ohjelmistotuotetta kestävyyden näkökulmasta. Lisäksi myös kehitysprosessin itsessään tulee olla ympäristöystävällinen (Naumann ym., 2011).

Nämä kaksi näkökulmaa kiteytyvät Naumannin (2011) esittämässä toisessa määritelmässä: ”Vihreä ja kestävä ohjelmistotuotanto (Green and Sustainable Software Engineering) on taitoa kehittää vihreää ja kestävää ohjelmistoa vihreällä ja kestävällä ohjelmistotuotantoprosessilla. Siksi se on taitoa määritellä ja kehittää ohjelmistotuotteita siten, että kielteiset ja myönteiset vaikutukset kestävään kehitykseen, jotka syntyvät ja/tai joiden odotetaan syntyvän ohjelmistotuotteesta koko sen elinkaaren aikana, arvioidaan, dokumentoidaan ja hyödynnetään jatkuvasti ohjelmistotuotteen jatko-optimoinnissa”



Kuvio 9. Esimerkki kestävästä ohjelmistokehitysprosessista, jossa kestävyyskatselmointi ja ennakkoarviointi ovat osa toteutusta Naumann (2011) mukaillen.

Tämä Naumann (2011) esimerkki (kuvio 9) ei toteuta täydellistä ohjelmistokehitysprosessia. Sen sijaan se ehdottaa parannuksia olemassa oleviin ohjelmistokehitysprosesseihin, jotta sidosryhmät voivat tunnistaa sekä ohjelmiston kehittämisestä että sen käytöstä syntyviä kestävyyteen liittyviä vaikutuksia. Ehdotettuja parannuksia ovat kestävyyskatselmoinnit ja -ennakkoarvioinnit, prosessiarviointi, kestävyyspäiväkirja sekä niin sanottu kestävyysretrospektiivi. Nämä parannukset muodostavat periaatteessa jatkuvan parantamisen syklin, jossa tarkastellaan järjestelmällisesti kestävyyteen liittyviä kysymyksiä. Prosessiarvioinnin tehtävänä on optimoida ohjelmiston ”tuotantoprosessin” kestävyyttä, kun taas kestävyyskatselmoinnit ja -ennakkoarvioinnit tukevat kehitettävän ohjelmistotuotteen kestävyyden optimointia. Näitä näkökulmia yhdistää kestävyysretrospektiivi, jonka avulla ohjelmistotuotteen koko elinkaaren aikaiset vaikutukset voidaan lopulta ottaa kattavasti huomioon. (Naumann, 2011.)

GREENSOFT-malli hyödyntää ICT:n vaikutustasojen tarkastelua ohjelmistokehityksen kontekstissa (Naumann ym., 2011). Mallissa elinkaarinäkökulma kohdistuu siihen, missä ohjelmiston elinkaaren vaiheessa nämä vaikutukset ilmenevät ja ovat tunnistettavissa.

Vihreän ja kestävän ohjelmiston epäsuorat kriteerit ja mittarit kohdistuvat ohjelmistotuotteen toisen ja kolmannen asteen vaikutuksiin. Niiden tunnistaminen ja mittaaminen on haastavaa, mutta rajatun käyttötarkoituksen omaavien järjestelmien vaikutuksia voidaan arvioida ainakin

likimääräisesti. Erityisesti sosiaalisten vaikutusten yleispätevä mittaaminen on vaikeaa, koska ne riippuvat ohjelmiston käyttötarkoituksesta ja sovellusalueesta. Kestävä ohjelmistokehitys edellyttää siksi, että suunnittelussa huomioidaan ohjelmistotuotteen koko elinkaari sekä sen kehittämisen konteksti ja tuotanto-olosuhteet (Naumann, 2011).

Microsoftin teknologiajohtaja Nathan Myhrvold esitti vuonna 1997 neljä ohjelmistolakia, joiden mukaan ohjelmistojen resurssivaatimukset kasvavat aina käytettävissä olevan laitteiston mukana ja uusi laitteistokehitys tapahtuu ohjelmistojen lisääntyvien vaatimusten seurauksena (Myhrvold, 1997). Tämä ennustaa sitä, että laitteiston suorituskyvyn paraneminen ei koskaan yksin ratkaise kasvavaa resurssitarvetta. Ainoa keino rajoittaa tätä kehityssuuntaa on keskittyä ohjelmistojen optimointiin ja niiden laitteistovaatimusten hallintaan.

Watson ym. (2008) käyttävät käsitettä Green IS kuvaamaan laajaa näkökulmaa, jossa tietojärjestelmiä sovelletaan kestävyyden tukemiseen koko taloudessa. Tähän sisältyy toimialojen tehokkuuden parantaminen erityisesti niillä sektoreilla, jotka aiheuttavat huomattavia kasvihuonekaasupäästöjä, kuten liikenne-, teollisuus- ja energia-aloilla. Green IS -näkökulmassa tietojärjestelmät nähdään keskeisenä keinona ympäristöongelmien ratkaisemisessa. Lopulta IT:n rooli tulisi ymmärtää sekä ongelman lähteenä että ratkaisun mahdollistajana

2.2.3 Kestävän ohjelmistokehityksen vaiheet

Penzenstadler ym. (2018) kehittivät teorian, jossa vipuvaikutuskohdat (leverage points, myöhemmin LP) ovat pisteitä, joissa pieni muutos ohjelmistossa voi johtaa merkittäviin koko järjestelmän laajuisiin muutoksiin. Artikkeli ehdottaa, että LP-kohtien avulla ohjelmistosuunnittelijat voivat tarjota näkemyksiä muutosmekanismeista tai eri tavoista löytää vaihtoisia ratkaisuja. LP on analyysityökalu, joka auttaa ammattilaisia tunnistamaan elementtejä, jotka voivat saada aikaan tehokkaita muutoksia ohjelmistojärjestelmän eri tasoilla. Penzenstadler ym. (2018) esittävät LP-ajattelun soveltamiseksi ohjelmistosuunnittelussa kahden kysymyksen mallia; ”Rakennammeko oikean järjestelmän?” ja ”Rakennammeko järjestelmän oikein?”. Kuten aluksi totesimme, kestävän ohjelmistokehityksen näkökulmasta ”oikea järjestelmä” tarkoittaa ratkaisua, joka tukee myös laajempia ympäristöön, yhteiskuntaan ja talouteen kohdistuvia kestävyyspyrkimyksiä.

Sosio-tekniset järjestelmät tarkoittavat alun perin järjestelmiä, joihin liittyy monimutkainen vuorovaikutus ihmisten, koneiden ja työjärjestelmän ympäristönäkökohtien välillä (Tieteen termipankki). Vipuvaikutuskohdat tarjoavat ohjelmistosuunnittelijoille mahdollisuuden tunnistaa ne järjestelmän kohdat, joissa pienet muutokset voivat tuottaa merkittäviä kestävyyshyötyjä.

Yhteistyö ohjelmistosuunnittelijan ja toimialan asiantuntijoiden välillä on keskeistä, jotta saadaan toteutettua kohde- ja matalasuhdekaavioita ja pystytään pitämään mielessä eri vipupisteiden tehokkuus vaatimusmäärittelyanalyysin aikana: Mihin kohteisiin ja mataliin tasoihin järjestelmä vaikuttaa (esim. energia, luonnonvarat, tavarantoimitus)? Voidaanko ohjelmistolla vakauttaa niitä (esim. optimoimalla puskureita, lyhentämällä määrien säätämisen viiveitä, tekemällä järjestelmän tilan tunnetuksi tai valvomalla ja säätämällä parametreja)? (Penzenstadler ym., 2018)

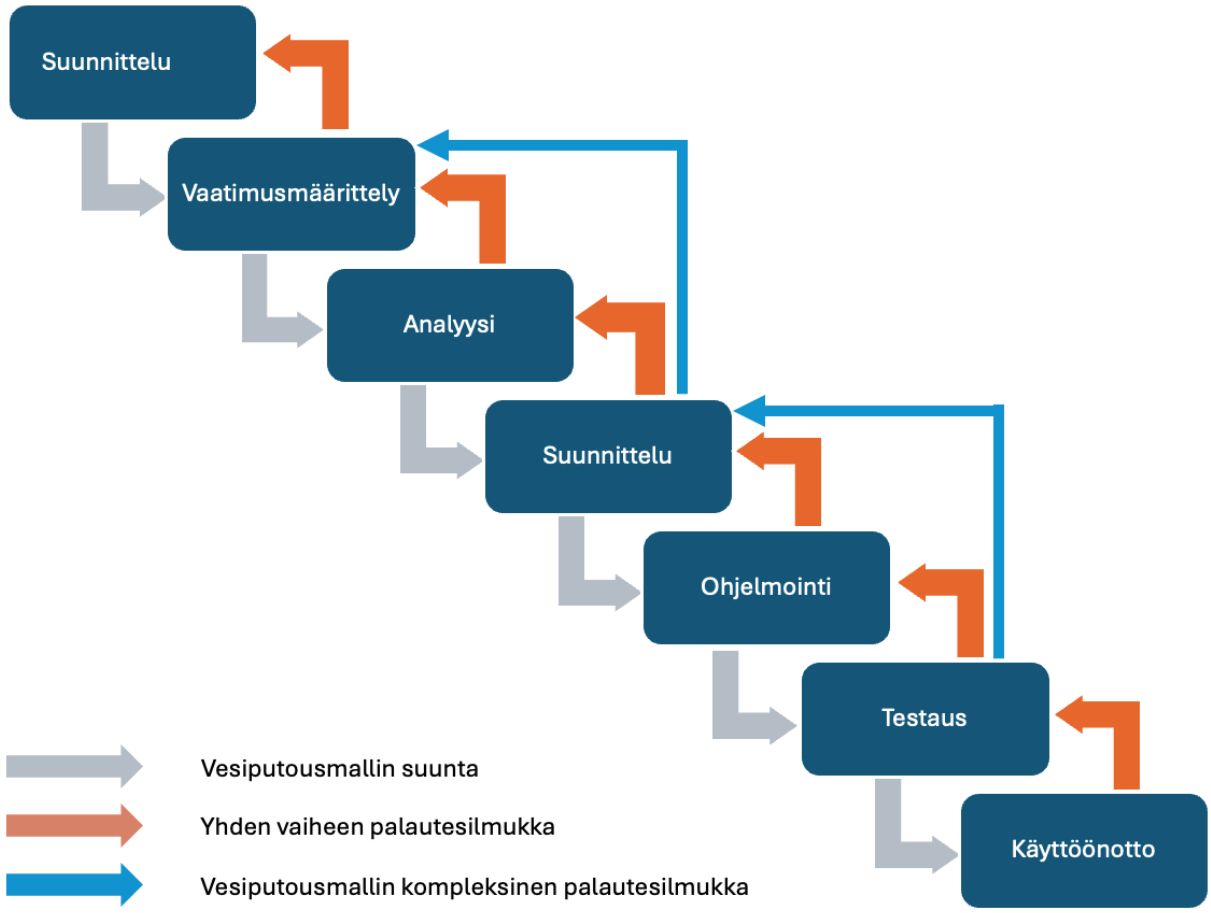
2.2.4 Vesiputousmalli

Vesiputousmalli on perinteinen, lineaarinen ohjelmistokehitysmalli, jossa työ etenee ennalta määriteltyjen vaiheiden kautta selkeässä järjestyksessä. Mallin perusajatus on, että kukin vaihe saatetaan päätökseen ennen seuraavaan siirtymistä, jolloin eteneminen on pääosin yksisuuntaista ja taaksepäin vaiheisiin palaaminen on rajattua sekä usein kustannuksiltaan kallista. Vesiputousmalli soveltuu erityisesti tilanteisiin, joissa vaatimukset ovat alusta asti suhteellisen vakaita, toimintaympäristöä ohjaa vahva sääntely tai halutaan vahva dokumentaatio ja vaiheportit.

Ruparelia (2010) tarkastelee keskeisiä ohjelmistokehityksen elinkaarimalleja ja niiden historiallista kehitystä. Vesiputousmalli, joka tunnetaan myös kaskadimallina, dokumentoitiin ensimmäisen kerran Beningtonin vuonna 1956 esittämässä mallissa ja myöhemmin Roycen vuonna 1970 julkaisemassa analyysissä. Malli on vaikuttanut merkittävästi ohjelmistokehityksen menetelmien kehittymiseen, koska se korosti systemaattista vaiheittaista etenemistä, jossa vaatimusten määrittely ja analyysi suoritetaan ennen suunnittelua ja toteutusta.

Beningtonin alkuperäisessä mallissa ohjelmistokehitys eteni vaiheittain sisältäen toiminnallisen analyysin, toiminnallisen määrittelyn ja koodausmäärittelyt. Koska jokaisen vaiheen päätteeksi muodostettiin dokumentoitu lähtötaso, Royce tunnisti riskin siinä, että suunnitteluongelmat voivat ilmetä vasta myöhemmissä kehitysvaiheissa. Tämän vuoksi hän

ehdotti malliin palautesilmukoita, joiden avulla kehitysprosessissa voidaan palata aiempiin vaiheisiin ja tarkentaa vaatimuksia tai suunnitteluratkaisuja.



Kuvio 10. Vesiputousmalli Roycen iteratiivisella palautteella Rupareliaa (2010) mukaillen.

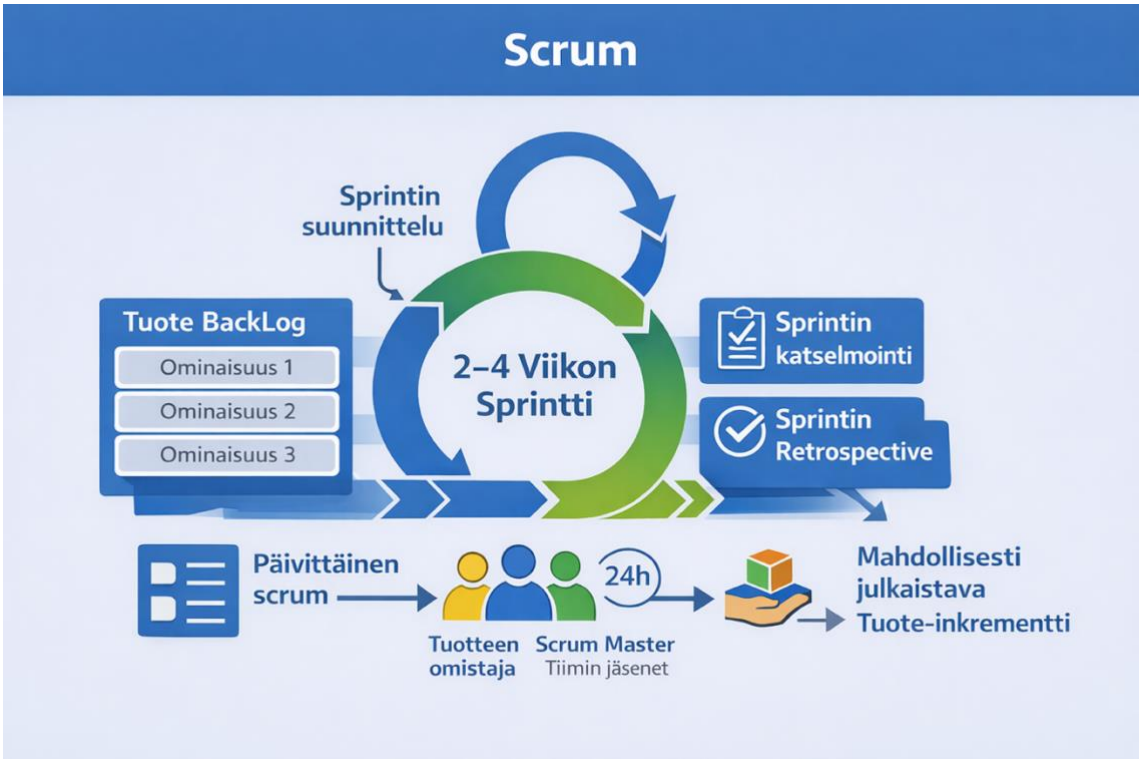
Tämä iterointi on esitetty kuviossa 10 oransseilla nuolilla, jotka kuvaavat kehitysprosessin palautesilmukoita. Näiden avulla voidaan palata aiempiin vaiheisiin ja tarkentaa suunnittelua kehitystyön edetessä. Royce kuitenkin korosti, että iterointi pelkästään peräkkäisten vaiheiden välillä ei välttämättä riitä, koska suunnittelu- ja toteutusvaiheissa voi ilmetä ongelmia, jotka edellyttävät palaamista useita vaiheita taaksepäin (siniset nuolet).

Esimerkiksi arviointi- tai testausvaiheen lopussa voidaan joutua palaamaan suoraan suunnitteluvaiheeseen ohittaen kehitysvaiheen. Vastaavasti suunnitteluvaiheen lopussa voidaan joutua palaamaan vaatimusten määrittelyvaiheeseen analyysivaiheen yli. Tällaisissa tilanteissa järjestelmän vaatimuksia ja suunnitteluratkaisuja tarkennetaan uudelleen testauksen ja suunnittelun aikana saatujen havaintojen perusteella.

2.2.5 Ketterä ohjelmistokehitys

Viimeisten kahden vuosikymmenen aikana ketterät ohjelmistokehitysmallit (Agile Software Development, ASD) ovat saavuttaneet huomattavaa suosiota ja tulleet laajasti käyttöön maailmanlaajuisessa IT-sektorilla (Lauesen, 2020). Ketterä kehitys korostaa toimivaa ohjelmistoa, iteratiivista ja inkrementaalista etenemistä sekä kykyä mukautua nopeasti muuttuviin vaatimuksiin. Vaikka dokumentaatiota usein kevennetään, sidosryhmien osallistaminen sekä vaatimusten tunnistaminen, priorisointi ja hallinta nähdään edelleen keskeisinä ketterän kehityksen elementteinä (Dreesen ym., 2019).

Toteutuksessa eniten käytetyt ketterät viitekehykset ovat Scrum, Extreme Programming ja Kanban, joista Scrum on käytetyin. Scrumissa (kuvio 11) on tapahtumia, joiden tulisi toteutua jokaisessa sprintissä. Nämä tapahtumat ovat sprintin suunnittelu (Sprint Planning), päivittäinen scrum (Daily Scrum), sprintin katselmointi (Sprint Review) ja sprintin retrospektiivi (Sprint Retrospective). Suunnittelua käytetään määrittämään, mitä työtä sprintin aikana tehdään. Päivittäinen scrum on päivittäin pidettävä tilannekatsaus, jossa viestitään, mitä kukin kehitystiimin jäsen tekee.



Kuvio 11. Scrum-prosessin perusrakenne. (GPT5.5)

Katselmointia käytetään keskustelemaan sprintin aikana saavutetuista asioista keskeisten sidosryhmien kanssa. Retrospektiivissä keskustellaan siitä, miten sprintti sujui ja mitä oppeja voidaan tunnistaa ja ottaa käyttöön seuraavassa sprintissä. Scrum-prosessin perusrakenne ja sprinttiin perustuva iteratiivinen eteneminen on esitetty kuviossa 11.

Ketterä lähestymistapa eroaa perinteisistä, suunnitelmavetoisista kehitysmalleista arvojensa ja käytäntöjensä osalta erityisesti vaatimustenhallinnassa. Ketterässä kehityksessä vaatimuksia ei tyypillisesti määritellä kokonaisuudessaan projektin alkuvaiheessa, vaan niitä täsmennetään ja priorisoidaan kehityksen edetessä. Näiden erojen seurauksena on syntynyt käsite ketterä vaatimustenhallinta (Agile Requirements Engineering, ARE) (Leffingwell, 2010). ARE:n on raportoitu vähentävän useita vaatimuksiin liittyviä ongelmia, kuten haasteita muuttuvien vaatimusten hallinnassa. Samanaikaisesti ketterä kehitys tuo mukanaan uusia haasteita, mikä korostaa ohjelmistokehitysprosessien jatkuvan parantamisen tarvetta (Heikkilä ym., 2015).

2.2.6 Ohjelmiston suunnittelu

Suunnitteluvaiheessa vaatimukset käännetään teknisiksi ratkaisuiksi, jotka määrittävät järjestelmän rakenteen, arkkitehtuurin ja tiedonkulun. Tämän vaiheen tavoitteena on varmistaa, että järjestelmä tukee sekä toiminnallisia että ei-toiminnallisia vaatimuksia, mukaan lukien suorituskyky, luotettavuus, turvallisuus ja ylläpidettävyys (Bourque & Fairley, 2014). Kestävän ohjelmistokehityksen näkökulmasta suunnitteluvaiheessa tehdään ratkaisevia valintoja, jotka vaikuttavat järjestelmän energiatehokkuuteen, resurssien käyttöön ja elinkaaren aikaiseen kestävyyteen. Arkkitehtuuriratkaisut, algoritmien tehokkuus ja järjestelmän skaalautuvuus voivat siten määrittää merkittävästi ohjelmiston ympäristövaikutuksia koko sen elinkaaren aikana (Naumann ym., 2011; Venters ym., 2018).

Ohjelmiston suunnittelussa kestävyyteen liittyy järjestelmän myöhempi muunneltavuus ja riippuvuuksien hallinta. Celkee Oy:n ym. (2013) selvityksen mukaan noin kolmannes tilaajista ei varmista järjestelmän arkkitehtuurin mahdollistavan toimittajan vaihtamista myöhemmässä vaiheessa. Tämä lisää toimittajalukkiutumisen riskiä ja voi rajoittaa järjestelmän myöhempää kehittämistä sekä ylläpitoa ja uudelleen kilpailuttamista. Kestävyyden näkökulmasta arkkitehtuurin tulisi tukea paitsi nykyisiä vaatimuksia myös järjestelmän pitkän aikavälin hallittavuutta.

2.2.7 Vaatimusten määrittely

Oakland (1989) käsittelee kokonaisvaltaisen laadunhallinnan näkökulmasta laadun, vaatimusten ja mittaamisen välistä suhdetta. Laadun käsite kytkeytyy läheisesti vaatimusten täyttämiseen. Yksinkertaisimmillaan laatu voidaan määritellä asiakkaan vaatimusten täyttämiseksi. Useat laadunhallinnan keskeiset tutkijat ja standardit ovat kuvanneet tätä näkökulmaa hieman eri tavoin.

Oaklandin (2014, s. 4–5) mukaan laadun määrittelemisessä on tärkeää asiakkaan vaatimusten ja tarpeiden tunnistaminen. Tämän tutkimuksen kannalta on keskeistä havaita, että laatu ei synny vasta toteutusvaiheessa, vaan se edellyttää vaatimusten tunnistamista ja täsmentämistä sekä suunnitteluun että toteutukseen. Luotettavuus on laadun ohella keskeinen tekijä tuotteiden ja palveluiden arvioinnissa, sillä se vaikuttaa merkittävästi asiakkaiden ostopäätöksiin vaihtoehtoja vertailtaessa. Tästä näkökulmasta vaatimusten tunnistaminen ja täsmällinen määrittely ovat keskeisessä roolissa minkä tahansa tuotteen tai palvelun laadun varmistamisessa.

Kokonaisvaltaisen laadunhallinnan (Total Quality Management, TQM) keskeinen ajatus on sisäisten ja ulkoisten asiakas–toimittaja suhteiden tunnistaminen organisaation sisällä. Jokaisella organisaation jäsenellä on omat asiakkaansa, jotka voivat olla joko ulkoisia loppukäyttäjiä tai organisaation sisäisiä sidosryhmiä, joille tuotetaan tietoa, palveluita tai järjestelmän osia. Tämän ajattelun mukaisesti laadun ja motivaation välinen yhteys perustuu siihen, että organisaation jäsenet ymmärtävät oman työnsä vaikutuksen asiakkaiden tarpeiden täyttämiseen. (Oakland, 2014, s.7–8.)

Suunnittelun laatu voidaan määritellä mittana sille, kuinka hyvin tuote tai palvelu on suunniteltu vastaamaan sovittuja vaatimuksia. Keskeinen tekijä laadun saavuttamisessa on selkeä spesifikaatio. Spesifikaatioiden laatiminen jokaisessa rajapinnassa auttaa tarkentamaan todellisia vaatimuksia sekä järjestelmän kyvykkyyksiä. Näin ollen spesifikaatioiden määrittely muodostaa ensimmäisen ja ratkaisevan vaiheen onnistuneessa kokonaislaadun toteuttamisessa (Oakland, 2014, s. 9–10.)

Vaatimusten määrittelyn merkitys nousi merkittäväksi huomioksi myös tietojärjestelmähankkeiden onnistumista koskevassa selvityksessä. Celkee Oy:n, Tietotekniikan liiton ja Ohjelmistoyrittäjien (2013) selvityksessä hankkeiden kriisiytymisen taustalla tunnistettiin useita haasteita, kuten kustannusarvion ylittyminen, aikataulun

pettäminen, erot projektin sisällöstä tulkinnasta, sekä tilaajan ja toimittajan väliset viestinnälliset puutteet. Lisäksi tilaajien ja toimittajien näkemykset määrittelyprosessin hallinnasta erosivat toisistaan. Tämä osoittaa sen, että vaatimusten määrittely ei ole vain tekninen vaihe, vaan myös projektinhallinnallinen ja organisatorinen kysymys. Kestävyyden näkökulmasta tämä on olennainen havainto, koska kestävyys ei voi muuttua ei-toiminnalliseksi vaatimukseksi, jos projektin tavoitteet ja vastuut ovat epäselviä.

Demingin Plan–Do–Check–Act (PDCA) -sykliä voidaan hyödyntää mittausjärjestelmien suunnittelussa ja laadun jatkuvassa parantamisessa. Ennen mittausjärjestelmien käyttöönottoa tulee kuitenkin vastata neljään keskeiseen kysymykseen: miksi mitataan, mitä mitataan, missä mittaaminen tapahtuu ja miten mittaaminen toteutetaan. Erityisesti kysymykseen ”miten mitata?” vastattaessa mittausjärjestelmän tulisi huomioida viisi keskeistä ulottuvuutta: vaikuttavuus, tehokkuus, tuottavuus, laatu ja vaikutus (Liker, 2004, s. 276–277; Oakland, 2014, s. 153)

2.2.8 Kestävyys ei-toiminnallisena vaatimuksena

Yksi keskeisimmistä ketterän vaatimustenhallinnan haasteista liittyy ei-toiminnallisiin vaatimuksiin (Non-Functional Requirements, NFR), joita kutsutaan myös laatuvaatimuksiksi (Heikkilä ym., 2015). Tutkimuksissa on toistuvasti raportoitu useita tähän liittyviä ongelmia. Näihin kuuluvat esimerkiksi ei-toiminnallisten vaatimusten sivuuttaminen toiminnallisiin vaatimuksiin keskittymisen vuoksi (Heikkilä ym., 2015), liian vähäinen dokumentaatio ei-toiminnallisten vaatimusten kuvaamiseksi (Ramesh ym., 2009; Inayat ym., 2015) sekä ketterien vaatimustenhallintatekniikoiden riittämättömyys NFR-vaatimusten käsittelyssä (Heikkilä ym., 2015; Wagner ym., 2018; Alsaqaf ym., 2017).

Näillä haasteilla on merkittäviä vaikutuksia ohjelmistojen hyväksyttävyyteen ja pitkäaikaiseen käyttöön. Useat esimerkit osoittavat, että riittämätön laatu johtaa järjestelmien hylkäämiseen käyttäjien toimesta (Charette, 2018). Ei-toiminnalliset vaatimukset muodostavat siten keskeisen mekanismin käyttäjien laatuodotusten ilmaisemisessa ja ovat keskeisiä ohjelmistojen hyväksyttävyyden, käytettävyyden ja luotettavuuden kannalta.

Vaatimustenhallinta on luonteeltaan kokonaisvaltainen prosessi, jonka tavoitteena on tunnistaa kaikki järjestelmään kohdistuvat vaatimukset sekä niiden keskinäiset riippuvuudet. Tämän vuoksi ei ole tarkoituksenmukaista suunnitella täysin erillistä vaatimustenhallintaprosessia yksittäiselle vaatimuskategorialle, kuten ei-toiminnallisille

vaatimuksille. Vaatimustenhallintaprosessia voidaan täydentää menetelmillä ja käytännöillä, jotka auttavat tunnistamaan, analysoimaan ja käsittelemään esimerkiksi kestävyyteen liittyviä vaatimuksia.

Ei-toiminnalliset vaatimukset, mukaan lukien kestävyyteen liittyvät vaatimukset, ovat usein sidosryhmille vähemmän ilmeisiä ja vaikeammin ilmaistavia esimerkiksi strukturoimattomissa haastatteluissa. Tämä voi johtaa vaatimusten puutteelliseen tunnistamiseen tai väärinymmärryksiin (Bourimi ym., 2010). Myös vaatimusten dokumentointiin liittyy rajoitteita. Usein käytetyt ketterät dokumentointimenetelmät, kuten käyttäjätarinat ja tarinakortit (Boehm ym., 2019), eivät sellaisenaan sovellu ei-toiminnallisten vaatimusten täsmälliseen esittämiseen. Näiden puutteiden korjaamiseksi on ehdotettu uusia tai mukautettuja dokumentointimenetelmiä, kuten W8 Story Cards -menetelmää (Farid ym., 2012). Kestävyyden tarkastelu ei-toiminnallisena vaatimuksena mahdollistaa kestävyyden systemaattisen huomioimisen ohjelmistokehityksen eri vaiheissa.

2.2.9 Lean-ajattelu kestävyyden ja vaatimustenhallinnan tukena

Lean-ajattelu on alun perin Toyotan tuotantojärjestelmästä (Toyota Production System, TPS) kehittynyt johtamis- ja kehittämisfilosofia, jonka keskeisenä tavoitteena on asiakasarvon maksimoiminen samalla kun hukkaa poistetaan prosesseista (Liker, 2004, s. 7, 16, 29–31). Lean-ajattelussa organisaation toimintaa tarkastellaan kokonaisvaltaisina prosesseina, joissa pyritään tunnistamaan ne työvaiheet, jotka tuottavat arvoa asiakkaalle, sekä poistamaan sellaiset toiminnot, jotka eivät lisää arvoa. Tämä lähestymistapa on laajentunut tuotannollisesta kontekstista myös palveluihin ja tietotyöhön, mukaan lukien ohjelmistokehitys (Liker, 2004, s. 14, 29, 33).

Lean-ajattelun keskeiset periaatteet ovat asiakasarvon tunnistaminen, arvovirran kuvaaminen, prosessien virtauttaminen, imuohjattu tuotanto sekä jatkuva parantaminen (Liker, 2004, s. 7). Näiden periaatteiden tavoitteena on tehdä organisaation toiminta läpinäkyväksi, vähentää turhaa työtä ja parantaa prosessien laatua ja tehokkuutta. Ohjelmistokehityksen kontekstissa Lean-ajattelun tavoitteena on erityisesti vähentää kehitysprosessin hukkaa, parantaa päätöksenteon laatua sekä lisätä asiakkaalle tuotettavaa arvoa (Poppendieck & Poppendieck ym., 2003).

Lean-ajattelu liittyy läheisesti kokonaisvaltaisen laadunhallinnan (Total Quality Management, TQM) perinteeseen, jossa keskeisiä periaatteita ovat jatkuva parantaminen, prosessien

systemaattinen kehittäminen, työntekijöiden osallistaminen sekä organisaation oppiminen (Oakland, 2014, s. 23–25). TQM-ajattelussa laadun määrittäjänä pidetään asiakasta, ja organisaation tehtävänä on kehittää prosessejaan siten, että asiakkaan tarpeet voidaan täyttää mahdollisimman tehokkaasti ja luotettavasti (Oakland, 2014, s. 4–5, 14–16). Kestävän kehityksen näkökulmasta tämä ajattelu voidaan laajentaa asiakaskeskeisyydestä kohti laajempaa sidosryhmäkeskeistä lähestymistapaa, jossa huomioidaan myös yhteiskunnan, ympäristön ja tulevien sukupolvien tarpeet (Oakland, 2014, s. 21; WCED, 1987).

Lean-ajattelun näkökulmasta keskeinen lähtökohta ohjelmistokehityksen parantamiselle on prosessien näkyväksi tekeminen. Prosessien kuvaaminen auttaa tunnistamaan työnkulkuun liittyviä päällekkäisyyksiä, pullonkauloja ja epäselviä vastuita sekä parantamaan resurssien käyttöä. Prosessien systemaattinen tarkastelu mahdollistaa myös organisaation toiminnan läpinäkyvyyden lisäämisen sekä kehittämistarpeiden tunnistamisen. (Poppendieck & Poppendieck, 2003). Prosessien kuvaamisen avulla voidaan esimerkiksi selkeyttää työnjakoa ja vastuita, tunnistaa osaamis- ja koulutustarpeita, arvioida resurssitarpeita sekä tukea muutosten hallintaa ja organisaation oppimista.

Ohjelmistokehityksessä prosessien kuvaaminen on erityisen tärkeää vaatimustenhallinnan näkökulmasta. Ei-toiminnalliset vaatimukset, kuten kestävyyteen liittyvät näkökulmat, jäävät helposti implisiittisiksi, mikäli järjestelmää ympäröiviä liiketoiminta- ja käyttöprosesseja ei ole mallinnettu riittävän täsmällisesti. Kun prosessit tehdään näkyviksi, voidaan paremmin tunnistaa ne kohdat, joissa ohjelmisto vaikuttaa organisaation toimintaan, resurssien käyttöön ja päätöksentekoon. Samalla voidaan tunnistaa kestävyyteen liittyviä vaatimuksia, kuten energiatehokkuutta, käytettävien resurssien optimointia tai käyttäytymiseen vaikuttavia mekanismeja. Tämän vuoksi tilaajan ja keskeisten sidosryhmien prosessien kuvaaminen ennen laatuattribuuttien, ei-toiminnallisten vaatimusten ja muiden järjestelmävaatimusten määrittelyä voi parantaa ohjelmistohankkeen lopputulosta sosiaalisesta, taloudellisesta ja ympäristöllisestä näkökulmasta.

Lean-ajattelussa keskeinen käsite on hukka, jolla tarkoitetaan kaikkia sellaisia toimintoja, jotka kuluttavat resursseja tuottamatta arvoa asiakkaalle. Toyotan tuotantojärjestelmässä tunnistetaan seitsemän keskeistä hukan tyyppiä: ylituotanto, odottelu, turha kuljettaminen, turha prosessointi, liikavarasto, turha liikkuminen sekä valmistusvirheet (Liker, 2004, s. 28–29). Ohjelmistokehityksen kontekstissa vastaavaa hukkaa ovat esimerkiksi epäselvät vaatimukset, tarpeeton uudelleenohjelmointi, virheiden korjaaminen myöhäisessä vaiheessa,

tarpeettomien ominaisuuksien kehittäminen tai kehitystiimin odottaminen puutteellisen määrittelyn vuoksi.

Hukan poistaminen liittyy myös kestävyyteen, sillä resurssien tehokkaampi käyttö vähentää sekä taloudellista että ympäristöllistä kuormitusta. Lean-ajattelun tavoitteena onkin tuottaa enemmän arvoa vähemmillä resursseilla, mikä on linjassa kestävän kehityksen periaatteiden kanssa. Resurssien tehokas käyttö, prosessien optimointi sekä jatkuva parantaminen voivat tukea sekä organisaation taloudellista suorituskykyä että ympäristövaikutusten vähentämistä. Sosiaalisesta näkökulmasta hukan tunnistaminen voi myös vähentää työn kuormittavuutta, koska epäselvät vastuut, päällekkäinen työ ja tarpeeton odottaminen lisäävät usein työn stressaavuutta.

Lean-ajattelu tarjoaa myös käytännön työkaluja prosessien analysointiin ja kehittämiseen. Näitä ovat esimerkiksi arvovirta-analyysi (Value Stream Mapping), jonka avulla voidaan mallintaa prosessien nykytila (AS-IS) ja tavoitetila (TO-BE), sekä Kanban, joka tekee työnkulun ja keskeneräisen työn näkyväksi kehitystiimille (Oakland, 2014, s. 312–317). Lisäksi Lean-ajatteluun liittyvät Gemba-kävelyt, joissa johto tarkastelee prosesseja siellä, missä työ todellisuudessa tapahtuu (Liker, 2004, s. 223–225), sekä A3-ongelmanratkaisumenetelmä, joka perustuu Demingin Plan–Do–Check–Act (PDCA) -parannussykliin (Liker, 2004, s. 244–248).

Ohjelmistokehityksessä Lean-ajattelun keskeinen hyöty liittyy erityisesti vaatimustenhallinnan alkuvaiheeseen. Kun organisaation keskeiset liiketoiminta- ja käyttöprosessit on kuvattu ja analysoitu, voidaan myös järjestelmään kohdistuvat ei-toiminnalliset vaatimukset tunnistaa johdonmukaisemmin. Tämä koskee erityisesti kestävyyteen liittyviä vaatimuksia, jotka liittyvät usein laajempien sosio-teknisten järjestelmien toimintaan eivätkä ole suoraan havaittavissa yksittäisen ohjelmistotoiminnon tasolla. Prosessien kuvaaminen voi tuoda esiin myös vipuvaikutuskohtia, riskejä ja tarpeettomia toimintoja, jotka muuten siirtyisivät sellaisinaan ohjelmiston vaatimuksiin ja myöhemmin sen käyttöön. Näin vaatimustenhallinnassa voidaan arvioida jo varhaisessa vaiheessa, tukeeko ohjelmisto nykyistä toimintatapaa vai auttaako se samalla parantamaan toimintaa sosiaalisesti, taloudellisesti ja ympäristöllisesti kestävämpään suuntaan.

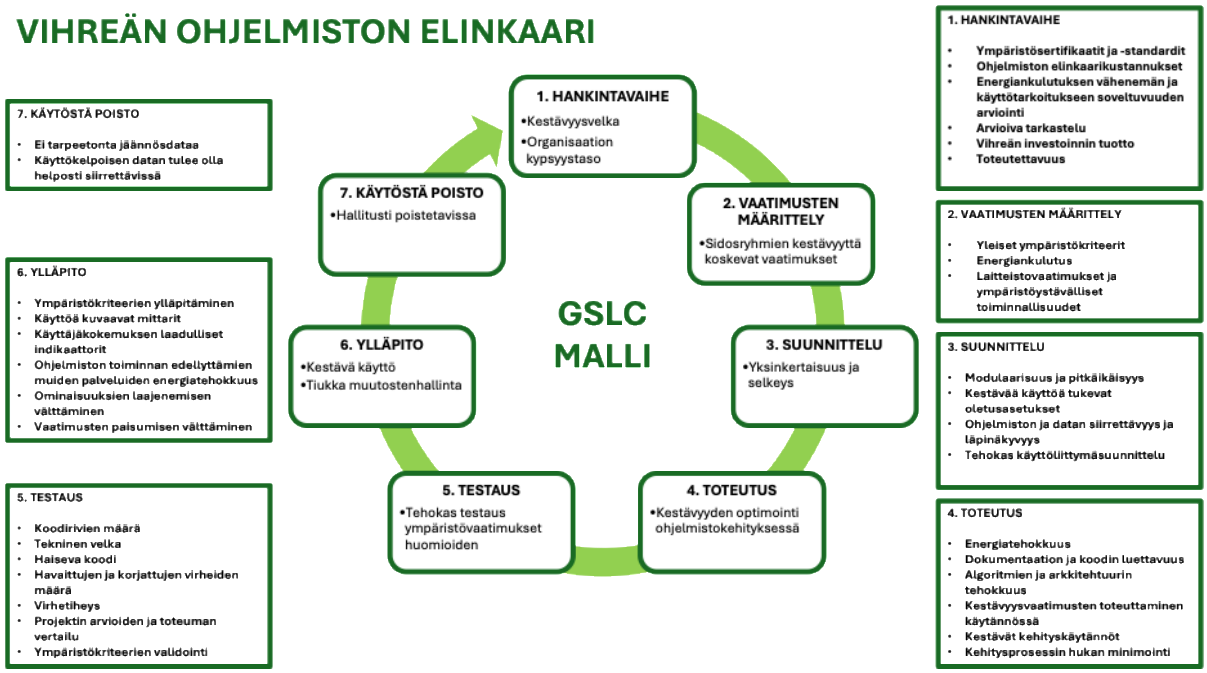
Lean-ajattelu ei kuitenkaan yksin ratkaise kestävyyden integrointia ohjelmistokehitykseen. Se ei esimerkiksi määritä kestävyyden ulottuvuuksia eikä tarjoa suoria mittareita kestävyyden arviointiin. Sen sijaan Lean tarjoaa menetelmällisen lähestymistavan, jonka avulla prosessit,

arvovirrat ja vaikutukset voidaan tehdä näkyviksi ennen varsinaista vaatimusmäärittelyä. Tämän ansiosta kestävyyteen liittyvät ei-toiminnalliset vaatimukset voidaan tunnistaa ja kuvata järjestelmällisemmin osana ohjelmistokehityksen päätöksentekoa.

2.2.10 Ohjelmiston elinkaari

Ohjelmistoprojekti toteutetaan vakiintuneiden vaiheiden mukaan. Software Engineering Body of Knowledge -opas (SWEBOK) sekä ISO/IEC/IEEE 12207:2017-standardi toimivat keskeisinä viitekehyksinä ohjelmistokehityksen elinkaariprosessien määrittelyssä. ISO/IEC/IEEE 12207:2017 määrittelee ohjelmiston elinkaariprosesseja, joita voidaan hyödyntää ohjelmistoprosessien määrittelyssä, ohjaamisessa ja kehittämisessä projektin tasolla (ISO/IEC/IEEE, 2017). SWEBOK:ssa kootaan yhteen vakiintuneet SDLC-vaiheet, jotka sisältävät vaatimusten määrittelyn, suunnittelun, toteutuksen, testauksen, käyttöönoton ja ylläpidon (Bourque & Fairley, 2014).

Ohjelmistokehityksen elinkaari (Software Development Life Cycle, SDLC) on käsitteellinen viitekehys, joka kuvaa ohjelmiston kehittämiseen liittyvien vaiheiden rakennetta ideasta käyttöönottoon ja ylläpitoon asti. SDLC-termiä käytetään toisinaan kuvaamaan sekä ohjelmiston että laajemman järjestelmän elinkaarta, sillä ohjelmistokehitys muodostaa keskeisen osan monien järjestelmien kehitysprosesseista.



Kuvio 12. Vihreän ohjelmiston elinkaari (GSLC)-malli. (Bourque & Fairley, 2014.)

SDLC-mallit tarjoavat rakenteellisen viitekehyksen ohjelmistokehityksen vaiheiden, niiden välisten riippuvuuksien sekä kehitysprosessin kokonaisuuden johdonmukaiseen jäsentämiseen. SDLC-mallit kuvaavat elinkaaren eri vaiheita ja niiden keskinäisiä suhteita, ja niitä käytetään jäsentämään kehitysprosessia alkaen toteutettavuustutkimuksesta aina järjestelmän käyttöönottoon ja ylläpitoon asti. (Ruparelia, 2010.)

Kuvio 12 havainnollistaa vihreän ohjelmiston elinkaari (GSLC)-mallia siten, että ympäristön kestävyyteen liittyvät tekijät sijoittuvat kuvan keskiosaan. Nämä tekijät määrittävät keskeiset menetelmälliset lähestymistavat, joiden avulla ympäristöllistä kestävyyttä pyritään edistämään ohjelmiston eri elinkaarivaiheissa. Kullekin vaiheelle ominaiset ympäristön kestävyyden mittarit ja indikaattorit on kuvattu erillisissä laatikoissa. (Naumann ym., 2011) Malli ja metodologia on erotettava toisistaan: malli kuvaa mitä vaiheita kehitysprosessiin kuuluu, kun taas metodologia kuvaa lisäksi, miten nämä vaiheet käytännössä toteutetaan (Ruparelia, 2010). Jotta ohjelmiston ideasta hautaan -eteneminen olisi kestävää, on keskeistä arvioida jo varhaisessa vaiheessa, onko kehityksen aloittaminen ylipäätään toteuttamiskelpoista. Mikäli kehityksen aikana syntyvät kustannukset ylittävät saavutettavissa olevat hyödyt, tulisi kehitystyö arvioida toteuttamiskelvottomaksi. Tämä korostaa erityisesti ohjelmistoprojektin alkuvaiheen merkitystä kestävyyden näkökulmasta (Naumann ym., 2011).

Ohjelmiston elinkaari ideasta hautaan (cradle-to-grave) voidaan kuvata kokonaisuutena, jossa järjestelmä syntyy tunnistetusta tarpeesta, kehittyy suunnittelun ja toteutuksen kautta aktiiviseen käyttöön ja lopulta poistuu hallitusti käytöstä. Elinkaaren alkuvaiheessa ideointi ja tarpeen tunnistaminen määrittävät, mikä ongelma tai mahdollisuus on kyseessä ja miksi ohjelmistoa ylipäätään tarvitaan. Tähän sisältyy myös harkinta siitä, onko ratkaisun toteuttaminen tarkoituksenmukaista vai olisiko perusteltua olla toteuttamatta sitä. Tätä seuraa esiselvitys ja kannattavuusarvio, jossa tarkastellaan liiketoimintahyötyjä, kustannuksia, riskejä, teknisiä reunaehtoja, aikataulua ja resursseja, minkä perusteella tehdään päätös hankkeen etenemisestä (Naumann ym., 2011).

SDLC-malleja luokitellaan kirjallisuudessa kolmeen pääryhmään: lineaarisiin, iteratiivisiin sekä näiden yhdistelmiin (Ruparelia, 2010). Lineaarisissa malleissa kehitys etenee vaiheittain siten, että yhden vaiheen valmistuminen johtaa seuraavan käynnistymiseen. Iteratiivisissa malleissa puolestaan kehitysvaiheisiin palataan toistuvasti, mikä mahdollistaa jatkuvan parantamisen ja vaatimusten tarkentamisen elinkaaren aikana. Yhdistelmämallit pyrkivät

hyödyntämään molempien lähestymistapojen vahvuuksia yhdistämällä vaiheittaisen etenemisen ja iteratiivisen kehityksen (Ruparelia, 2010).

Hankinta- ja käynnistysvaiheessa määritellään toteutustapa, kuten oma kehitys, ulkoinen toimittaja tai valmisratkaisun käyttöönotto. Tässä vaiheessa laaditaan myös sopimukset sekä rajataan projekti ja sen hallintamalli. Vaatimusmäärittelyvaiheessa täsmennetään, mitä järjestelmän tulee tehdä ja millaisia laatuominaisuuksia siltä edellytetään, esimerkiksi suorituskyvyn, tietoturvan, käytettävyyden ja energiatehokkuuden osalta, sekä määritellään hyväksymiskriteerit. Tässä vaiheessa määritellään myös ei-toiminnalliset vaatimukset, kuten kestävyyteen liittyvät laatuattribuutit. Suunnitteluvaiheessa nämä vaatimukset konkretisoidaan arkkitehtuuriksi ja teknisiksi ratkaisuiksi, kuten tietomalleiksi, integraatioiksi ja käyttöliittymiksi, siten että kokonaisuus tukee asetettuja tavoitteita.

Toteutusvaiheessa järjestelmä rakennetaan käytännössä koodaamalla, konfiguroimalla ja integroimalla komponentteja. Samalla huolehditaan versionhallinnasta ja peruslaadunvarmistuksesta. Testaus- ja verifiointivaiheessa varmistetaan, että ohjelmisto täyttää sille asetetut vaatimukset sekä toiminnallisesti että ei-toiminnallisesti, ja havaitut virheet korjataan. Käyttöönotossa järjestelmä siirretään tuotantoympäristöön, suoritetaan tarvittavat migraatiot, koulutukset ja valmistelut, ja ohjelmisto otetaan varsinaiseen käyttöön.

Operointi- ja ylläpitovaihe on tyypillisesti elinkaaren pisin vaihe. Tässä vaiheessa järjestelmää valvotaan, häiriöitä hallitaan, tehdään päivityksiä ja tietoturvakorjauksia sekä seurataan kapasiteettia, kustannuksia ja energiankulutusta jatkuvan parantamisen periaatteiden mukaisesti. Kestävän ohjelmistokehityksen kirjallisuudessa ylläpito- ja käyttövaihe korostuvat, koska ohjelmiston resurssien kulutus ja käytönaikaiset vaikutukset realisoituvat usein vasta pitkän käyttöjakson aikana (Naumann ym., 2011) Elinkaaren lopussa, käytöstäpoistovaiheessa ohjelmisto ajetaan hallitusti alas. Tähän sisältyy datan arkistointi tai siirto, integraatioiden purkaminen, käyttöoikeuksien sulkeminen, sopimusten päättäminen sekä infrastruktuurin vapauttaminen siten, ettei järjestelmästä jää hallitsemattomia jäänteitä käyttöön. Elinkaaren päättymisvaihe on tärkeä myös kestävyyden näkökulmasta, koska se vaikuttaa esimerkiksi datan hallintaan, resurssien vapauttamiseen ja infrastruktuurin energiankulutukseen (Naumann ym., 2011).

2.2.11 Customer Value -työkalut

Asiakasarvon tunnistaminen ja ymmärtäminen on keskeinen osa vaatimusten määrittelyä sekä tuotteiden ja palveluiden kehittämistä. Organisaatiot hyödyntävät erilaisia asiakaslähtöisiä analyysimenetelmiä tunnistaakseen ja priorisoidakseen käyttäjien tarpeita sekä muuntaakseen nämä tarpeet teknisiksi vaatimuksiksi. Asiakasarvon analysointi tukee vaatimustenhallintaa erityisesti tilanteissa, joissa järjestelmän tavoitteena on tuottaa sekä liiketoiminnallista että kestävyyteen liittyvää arvoa. Näitä menetelmiä voidaan tarkastella osana vaatimustenhallinnan prosessia, jossa sidosryhmien odotukset, tarpeet ja arvot pyritään systemaattisesti tunnistamaan ja kääntämään toteutettaviksi ratkaisuiksi.

2.2.12 Voice of Customer

Voice of Customer (VoC) viittaa menetelmälliseen lähestymistapaan, jonka avulla pyritään tunnistamaan ja priorisoimaan asiakkaiden tarpeita sekä ymmärtämään niiden merkitystä tuotteen tai palvelun kehittämisessä (Freeman & Radziwill, 2018). VoC-menetelmät yhdistävät laadullisia ja määrällisiä analyysimenetelmiä, joiden avulla voidaan tunnistaa sekä asiakkaiden suoraan ilmaisemia tarpeita että sellaisia latentteja tarpeita, jotka eivät ole helposti havaittavissa ilman analyyttista tarkastelua. Asiakasarvon analysointi tukee vaatimustenhallintaa erityisesti tilanteissa, joissa järjestelmän tavoitteena on tuottaa sekä liiketoiminnallista että kestävyyteen liittyvää arvoa.

Kokonaisvaltaisen laadunhallinnan näkökulmasta asiakkaan käsite ulottuu myös organisaation sisälle. Demingin mukaan jokainen prosessin vaihe voidaan nähdä asiakkaana seuraavalle vaiheelle, jolloin jokaiselle prosessille tulisi toimittaa täsmälleen se, mitä seuraava vaihe tarvitsee oikeaan aikaan. Tätä ajattelua tukee Demingin Plan–Do–Check–Act (PDCA) -malli, joka korostaa jatkuvaa parantamista organisaation prosesseissa. Jatkuva parantaminen tarkoittaa prosessien systemaattista kehittämistä päivittäisessä työssä sekä lisäarvoa tuottamattoman toiminnan vähentämistä (Oakland, 2014, s.154) PDCA-sykliä voidaan soveltaa myös ohjelmistokehityksessä jatkuvan parantamisen periaatteena.

VoC-menetelmät ovat keskeisiä erityisesti vaatimustenhallinnan varhaisessa vaiheessa, jossa järjestelmän arvoa tarkastellaan samanaikaisesti käyttäjä-, liiketoiminta- ja kestävyysperspektiivistä. VoC-prosessi voidaan kuvata nelivaiheisena kokonaisuutena, jossa asiakkaan tarpeet tunnistetaan, analysoidaan, muutetaan kehittämistä ohjaaviksi vaatimuksiksi ja kytketään jatkuvaan asiakasarvon arviointiin.

Prosessi voidaan esittää seuraavasti:

Kuvio 13. Voice of Customer -prosessin vaiheet vaatimustenhallinnassa.

2.2.13 Customer Utility Map

Customer Utility Map -menetelmää käytetään asiakasarvon analysointiin koko asiakaskokemuksen elinkaaren aikana (taulukko 1). Menetelmä perustuu taulukkomalliin, jossa asiakaskokemuksen eri vaiheet asetetaan vastakkain keskeisten hyötytekijöiden (utility levers) kanssa. Tämän lähestymistavan avulla voidaan tunnistaa, missä vaiheissa asiakasarvo syntyy ja missä vaiheissa sitä voidaan parantaa. Samalla menetelmä auttaa tunnistamaan mahdollisia kehityskohteita tuotteen tai palvelun suunnittelussa sekä tukee asiakaslähtöistä kehittämistä koko tuotteen tai palvelun elinkaaren aikana. (Freeman & Radziwill, 2018.)

	Ostaminen	Toimitus	Käyttö	Lisätarvikkeet/ täydennykset	Ylläpito	Käytöstä poisto
Tuottavuus						
Yksinkertaisuus						
Helppous						
Riski						
Mielikuva ja elämyksellisyys						
Ympäristöystävällisyys						

Taulukko 1. Customer Utility Map -työkalun malli.

Customer Utility Map -menetelmä auttaa analysoimaan asiakasarvon syntymistä tuotteen tai palvelun elinkaaren eri vaiheissa. Menetelmässä asiakaskokemuksen vaiheet, kuten ostaminen, toimitus, käyttö, ylläpito ja käytöstä poisto, asetetaan vastakkain keskeisten hyötytekijöiden kanssa. Näin voidaan tunnistaa, missä vaiheissa asiakas kokee suurinta arvoa ja missä puolestaan esiintyy kehityspotentiaalia. Menetelmä tukee vaatimusten määrittelyä tunnistamalla, missä vaiheissa järjestelmä tuottaa eniten arvoa käyttäjälle.

2.2.14 Ohjelmiston toteutus

Toteutusvaiheessa ohjelmiston suunnitteluratkaisut viedään käytäntöön ohjelmointityön avulla. Tässä vaiheessa kehitetään järjestelmän komponentit, toteutetaan integraatiot ja rakennetaan tietorakenteet siten, että ne vastaavat suunnitteluvaiheessa määriteltyjä arkkitehtuuriratkaisuja. Toteutusvaiheella on keskeinen merkitys kestävän ohjelmistokehityksen kannalta, sillä siinä tehtävät ratkaisut vaikuttavat suoraan ohjelmiston energiatehokkuuteen, resurssien kulutukseen ja suorituskykyyn. Algoritmien tehokkuus, koodin optimointi ja resurssien tarkoituksenmukainen käyttö voivat pienentää ohjelmiston energiankulutusta ja parantaa sen kestävyyttä (Naumann ym., 2011)

Testausvaiheessa perinteisesti tärkein tavoite on ohjelmistovirheiden tunnistaminen ja korjaaminen sekä sen varmistaminen, että ohjelmisto täyttää toiminnalliset vaatimukset. Kestävän ohjelmistokehityksen näkökulmasta testausvaiheessa tulisi kuitenkin arvioida myös ohjelmiston ympäristövaikutuksia ja energiatehokkuutta. Tässä vaiheessa voidaan validoida

ohjelmiston ympäristövaatimusten täyttyminen sekä arvioida ohjelmiston ”vihreyttä” esimerkiksi suorituskyvyn ja resurssien käytön näkökulmasta. Standardien ja sertifiointikriteerien noudattamisen validointi lisää niiden uskottavuutta ja varmistaa, että ohjelmistojen kehityksessä huomioidaan tehokkuus aidosti. (Kivimäki ym., 2024)

Ohjelmiston ylläpito- ja käyttövaiheessa on keskeistä varmistaa, että alkuperäiset vaatimukset täyttyvät mahdollisimman hyvin. Ylläpito- ja käyttövaihe on tyypillisesti ohjelmiston elinkaaren pisin vaihe, ja sen ympäristövaikutus on usein merkittävin. Päivitysten maltillinen toteuttaminen on tärkeää, sillä jatkuvat päivitykset ja uudet toiminnallisuudet voivat lisätä laitteiston suorituskykyvaatimuksia ja lyhentää laitteiston käyttöikää. Laitteiston toiminnallisen käyttöiän maksimoiminen on siten keskeinen kestävyyttä edistävä tekijä.

Kestävää käyttöä edistääkseen ohjelmistot tulisi suunnitella ympäristöystävällisillä oletusasetuksilla ja toiminnoilla. Lisäksi käyttäjille tulisi tarjota ohjeistusta ohjelmiston tehokkaaseen käyttöön. Käyttäjien tietoisuuden lisääminen tehokkaista käyttötavoista on tärkeä osa vihreää digitalisaatiota, sillä loppukäyttäjillä ei usein ole riittäviä työkaluja oman digitaalisen hiilijalanjälkensä pienentämiseen. (Kivimäki ym., 2024)

2.3 Tutkimuksen teoreettinen viitekehys

Edellisissä luvuissa tarkasteltiin kestävää kehitystä, kestävää projektinhallintaa, kestävää ohjelmistokehitystä sekä ohjelmistojen vaikutuksia laajempiin sosio-teknisiin järjestelmiin. Lisäksi tarkasteltiin ohjelmistokehityksen elinkaarimalleja, ei-toiminnallisia vaatimuksia sekä Lean-ajattelun roolia prosessien kehittämisessä. Näiden teoreettisten näkökulmien perusteella tässä tutkimuksessa muodostetaan viitekehys, jonka avulla kestävyyden huomioimista ohjelmistokehityksessä voidaan analysoida systemaattisesti. Tutkimuksen teoreettinen viitekehys perustuu kolmeen toisiaan täydentävään näkökulmaan: kestävyyden ulottuvuuksiin, ohjelmistojen vaikutustasoihin sekä ohjelmistokehityksen elinkaarivaiheisiin.

Ensimmäinen näkökulma liittyy kestävyyden ulottuvuuksiin. Kestävyys on kirjallisuudessa perinteisesti jäsennetty kolmeen osa-alueeseen: ympäristölliseen, taloudelliseen ja sosiaaliseen kestävyyteen (Elkington, 1997). Ohjelmistokehityksen kontekstissa tätä mallia on kuitenkin laajennettu kattamaan myös tekninen ja yksilöllinen ulottuvuus (Penzenstadler ym., 2014; Becker ym. 2015). Tekninen kestävyys viittaa ohjelmistojen kykyyn säilyttää toiminnallisuutensa ja ylläpidettävyytensä pitkällä aikavälillä, kun taas yksilöllinen ulottuvuus

liittyy käyttäjien hyvinvointiin, autonomiaa tukeviin järjestelmiin sekä työn laatuun. Näiden ulottuvuuksien avulla voidaan tarkastella ohjelmistojen kestävyyttä kokonaisvaltaisesti.

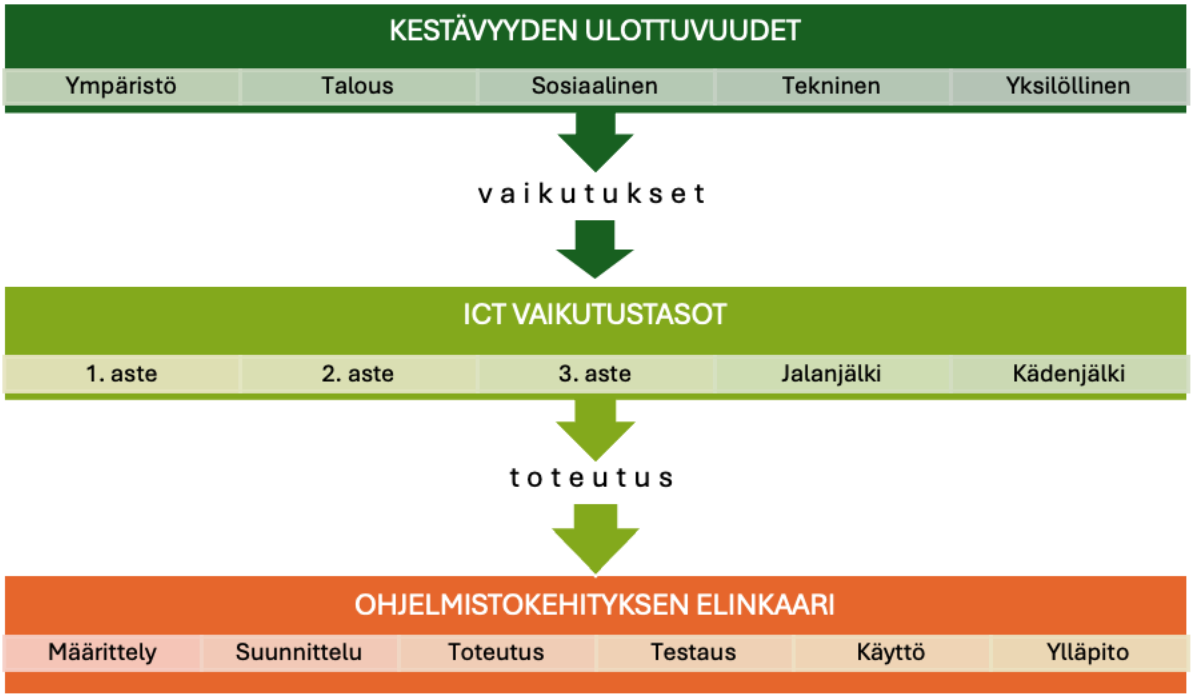
Toisena näkökulmana ovat ohjelmistojen vaikutustasot. ICT:n kestävyyttä koskevassa tutkimuksessa vaikutukset jaotellaan usein kolmeen tasoon: ensimmäisen, toisen ja kolmannen asteen vaikutuksiin (Hilty & Aebischer, 2015). Ensimmäisen asteen vaikutukset liittyvät ICT-järjestelmien suoriin ympäristövaikutuksiin, kuten energiankulutukseen, infrastruktuuriin ja laitteistojen elinkaareen. Toisen asteen vaikutukset syntyvät ICT:n käytön mahdollistamista prosessimuutoksista, esimerkiksi toimintojen tehostumisesta tai kulutuskäyttäytymisen muutoksista. Kolmannen asteen vaikutukset puolestaan ilmenevät laajempina systeemisinä muutoksina yhteiskunnassa, taloudessa ja käyttäytymisessä pitkällä aikavälillä. Näiden vaikutustasojen tarkastelu on keskeistä, koska ohjelmistojen merkittävimmät kestävyyteen liittyvät vaikutukset syntyvät usein epäsuorasti niiden käytön kautta.

Kolmantena näkökulma ovat ohjelmistokehityksen elinkaaren vaiheet. Ohjelmistokehityksen elinkaari (Software Development Life Cycle, SDLC) tarjoaa rakenteellisen viitekehyksen ohjelmistokehityksen vaiheiden jäsentämiseen ideasta käyttöönottoon ja ylläpitoon saakka (Ruparelia, 2010). Elinkaariajattelun näkökulmasta kestävyyden huomioiminen ei rajoitu yksittäiseen kehitysvaiheeseen, vaan se tulee integroida koko kehitysprosessiin. Keskeisiä vaiheita ovat erityisesti vaatimusten määrittely, suunnittelu, toteutus, testaus, käyttöönotto sekä järjestelmän ylläpito ja käytöstä poistaminen. Kestävyyteen liittyvät vaikutukset realisoituvat usein vasta järjestelmän käyttövaiheessa, minkä vuoksi elinkaarinäkökulma on keskeinen kestävyyden arvioinnissa. Ohjelmistokehityksen näkökulmasta kestävyys voidaan ymmärtää erityisesti ei-toiminnallisena vaatimuksena eli ohjelmiston laatuattribuuttina. Ei-toiminnalliset vaatimukset kuvaavat järjestelmän ominaisuuksia, kuten suorituskykyä, tietoturvaa, käytettävyyttä ja luotettavuutta. Kestävyyden tarkasteleminen laatuattribuuttina mahdollistaa sen, että kestävyyteen liittyvät tavoitteet voidaan sisällyttää järjestelmällisesti ohjelmiston suunnitteluun, arkkitehtuuriin ja toteutukseen. Tämä tarkoittaa, että kestävyyttä koskevat vaatimukset voidaan määritellä, dokumentoida ja arvioida samalla tavoin kuin muitakin järjestelmän laatuvaatimuksia.

Lean-ajattelu tukee tätä lähestymistapaa erityisesti vaatimustenhallinnan alkuvaiheessa. Lean-ajattelun keskeinen tavoite on tehdä prosessit, arvovirrat ja resurssien käyttö näkyviksi sekä tunnistaa niihin liittyvä hukka (Poppendieck & Poppendieck, 2003). Kun organisaation

keskeiset liiketoiminta- ja käyttöprosessit on kuvattu, voidaan myös ohjelmistoon kohdistuvat ei-toiminnalliset vaatimukset tunnistaa systemaattisemmin. Prosessien analysointi auttaa tunnistamaan esimerkiksi resurssien käyttöön, energiatehokkuuteen tai toiminnan optimointiin liittyviä kestävyyden näkökulmia jo ennen varsinaista ohjelmiston suunnittelua.

Tässä tutkimuksessa edellä esitetyt näkökulmat yhdistetään viitekehykseksi, jonka avulla voidaan tarkastella kestävyyden huomioimista ohjelmistokehityksessä. Viitekehyksen lähtökohtana on, että kestävyys voidaan ymmärtää ohjelmiston laatuattribuuttina, jonka vaikutukset ulottuvat useille kestävyyden ulottuvuuksille ja ilmenevät eri vaikutustasoilla. Samalla kestävyys tulee huomioida ohjelmistokehityksen eri vaiheissa, erityisesti vaatimustenhallinnan ja suunnittelun alkuvaiheessa, joissa keskeiset arkkitehtuuri- ja suunnitteluratkaisut tehdään.



Kuvio 14. Kestävyyden ulottuvuuksien, ICT-vaikutustasojen ja ohjelmistokehityksen elinkaaren muodostama tarkastelukehys kestävälle ohjelmistokehitykselle.

Kuviossa 14 viitekehys on jäsennetty kolmitasoisena kokonaisuutena. Ylimmällä tasolla ovat kestävyyden ulottuvuudet, joiden avulla arvioidaan, mihin kestävyyden osa-alueisiin ohjelmiston suunnittelu- ja kehitysratkaisut voivat vaikuttaa. Keskimmäisellä tasolla ovat ICT:n vaikutustasot, joiden avulla tarkastellaan, ilmenevätkö vaikutukset suorina

ensimmäisen asteen vaikutuksina, epäsuorina toisen asteen vaikutuksina vai laajempina systeemisinä kolmannen asteen vaikutuksina. Samalla tarkasteluun sisältyy ohjelmistojen hiilijalanjäljen ja hiilikädenjäljen välinen ero: ohjelmisto voi sekä kuluttaa resursseja että mahdollistaa kestävyyshyötyjä tai -haittoja muissa prosesseissa. Alimmalla tasolla on ohjelmistokehityksen elinkaari, joka osoittaa, missä kehityksen vaiheissa kestävyyteen liittyviä vaatimuksia, vaikutuksia ja päätöksiä tulee arvioida. Kuvion tarkoituksena on havainnollistaa, että kestävyys ei ole yksittäinen kehitysvaiheeseen sidottu lisävaatimus, vaan koko elinkaaren läpäisevä laatuattribuutti, jonka vaikutuksia tulee tarkastella sekä kestävyyden ulottuvuuksien että ICT-vaikutustasojen kautta.

Tätä viitekehystä hyödynnetään tutkimuksen empiirisessä osassa analysoitaessa, miten ohjelmistokehittäjät ja organisaatiot tunnistavat ja huomioivat kestävyyteen liittyviä vaatimuksia ohjelmistokehityksen eri vaiheissa sekä millaisia vaikutuksia ohjelmistojen suunnitteluratkaisuilla voi olla kestävyyden eri ulottuvuuksiin.

3 Aineisto ja menetelmät

Luvussa kuvataan tutkimuksen metodologinen lähestymistapa, empiirinen aineisto, haastateltavien ja organisaatioiden konteksti, aineistonkeruuseen liittyvät menetelmät ja valinnat, analyysimenetelmä sekä aineiston anonyymisointi ja rajaukset. Tutkimuksen empiirinen osuus perustuu puolistrukturoituihin asiantuntijahaastatteluihin. Näiden avulla tarkasteltiin kestävyyden huomioimista ohjelmistokehityksen käytännöissä, päätöksenteossa, vaatimustenhallinnassa, arkkitehtuurissa ja organisaatioiden toimintamalleissa, mikä on esitetty luvussa 4. Varsinaiset tutkimustulokset ja niiden teoreettinen tulkinta esitetään luvussa 5.

3.1 Laadullinen tutkimus

Laadullinen tutkimus on tutkimuksen metodologinen lähestymistapa, jonka tavoitteena on ymmärtää ja tulkita ilmiöitä niiden kontekstissaan ja tuottaa ilmiöstä teoreettisesti perusteltu selitys empiirisen aineiston pohjalta (Eskola ym., 1996, s. 61–62). Laadulliset menetelmät korostavat tiedon ymmärtämistä sekä sen tulkintaa, ja soveltuvat tutkimusasetelmiin, joissa tutkimuksen kohteena ovat monimutkaiset, sosio-tekniset käytännöt, merkityksen muodostuminen sekä toimintaympäristön vaikutus tutkittavaan ilmiöön (Eriksson ym., 2016, s. 4–5).

Tämän tutkimuksen ensisijaisena tavoitteena on muodostaa kokonaisvaltainen ymmärrys siitä, miten kestävyyden näkökulmat ja kädenjälkiajattelu voidaan käytännössä tuoda osaksi ohjelmistokehitystä, millaisia systeemisiä vaikutuksia heikko suunnittelu voi synnyttää ja miten organisaatiot voivat tunnistaa, mitata ja hallita näitä vaikutuksia. Laadullinen tutkimusote mahdollistaa ilmiön tarkastelun ongelman sisältäpäin, niin että huomio kohdistuu organisaatioiden käytänteisiin, prosesseihin, perusteluihin, artefakteihin ja päätöksentekoon. Laadullisessa tutkimuksessa pyritään kuvaamaan todellisuutta kokonaisvaltaisesti ja tuottamaan ymmärrystä tutkittavasta kohteesta sen omassa kontekstissaan (Hirsjärvi ym. 2007, s. 157).

Ohjelmistotekniikan empiirisessä tutkimuksessa laadulliset lähestymistavat ovat vakiintuneita, kun tutkimuksen kohteena ovat kehitysprosessit, organisaatiokäytännöt tai monimutkaiset järjestelmäkokonaisuudet. Tapaustutkimuksia koskevat ohjeistukset korostavat kontekstuaalisen ymmärryksen merkitystä sekä useiden aineistolähteiden hyödyntämistä ilmiön kokonaisvaltaisessa analyysissa (Runeson & Höst, 2009)

Empiirisen ohjelmistotekniikan menetelmäkirjallisuus korostaa, että tutkimusasetelman tulee olla johdonmukaisesti yhteydessä tutkimuskysymyksiin, tutkimuksen tavoitteeseen ja tarkasteltavan ilmiön luonteeseen. Menetelmää ei siten tule valita irrallisena tutkimusasetelmasta, vaan sen tulee tukea sitä, millaista tietoa tutkimuksessa pyritään tuottamaan (Runeson & Höst, 2009; Wohlin ym., 2012). Lisäksi laadulliset menetelmät mahdollistavat ohjelmistokehityksen käytäntöjen, kokemusten, vuorovaikutuksen ja päätöksenteon tarkastelun, mikä on keskeistä sosio-teknisten ilmiöiden analyysissä, joissa tekniset ratkaisut kytkeytyvät ihmisten, organisaatioiden ja työprosessien toimintaan (Seaman, 1999).

Tutkimusote nojaa lähestymistapaan, jossa ohjelmistojen kestävyys- ja hiilikädenjälkivaikutukset nähdään sosio-teknisenä ilmiönä. Tässä kontekstissa vaikutuksia syntyy sekä ohjelmiston teknisistä ratkaisuista sekä miten ohjelmisto suunnitellaan, toteutetaan, otetaan käyttöön ja liitetään osaksi laajempaa sosio-teknistä toimintaympäristöä. Tämän vuoksi tutkimus käsittelee mekanismeja ja käytäntöjä, joiden kautta vaikutukset syntyvät ja joiden avulla vaikutuksia voidaan hallita osana ohjelmistokehitysprosessia. Tutkimuskysymykset ohjaavat metodologisia valintoja niin, että aineistonkeruu ja analyysi rakentuvat niiden ympärille ja tavoitteena on menetelmällinen johdonmukaisuus (Berryman 2019, s. 278) Näin tutkimuskysymysten luonne määrittää millaista tietoa on tarpeen kerätä ja miten sitä analysoidaan.

Laadullisen analyysin tavoitteena on tunnistaa, miten kestävyys ja positiivinen hiilikädenjälki ilmenevät ohjelmistokehityksen eri vaiheissa sekä missä kohdissa voidaan tunnistaa keskeisiä vipupisteitä kestävyyden parantamiseksi. Analyysissä tarkastellaan myös, miten suunnitteluvirheet, puutteellinen vaatimusmäärittely tai heikot arkkitehtuuriratkaisut voivat synnyttää negatiivisia hiilijalanjälki- ja systeemivaikutuksia ohjelmiston elinkaaren aikana. Lisäksi analyysin avulla jäsennetään käytännön toimintatapoja vaikutusten tunnistamiseen, mittaamiseen ja hallintaan. Havaintojen pohjalta pyritään muodostamaan poikkitieteellinen viitekehys tai mittaristo, jota voidaan soveltaa ohjelmistokehitysprosesseissa.

Laadullinen tutkimustapa mahdollistaa ilmiön tarkastelun teknisten- ja organisatoristen näkökulmien kautta. Se tuottaa kontekstiin sidonnaisen kokonaiskuvan siitä, kuinka kestävyysvaatimukset ja vaikutusten arviointi voidaan saada osaksi ohjelmistokehitystä.

3.2 Haastateltavat ja niiden kuvaus

Tässä luvussa kuvataan tutkimukseen osallistuneet organisaatiot ja haastateltavat sekä perustellaan heidän valintansa suhteessa tutkimuksen tavoitteisiin. Laadullisessa tutkimuksessa aineiston valinta perustuu tarkoituksenmukaiseen otantaan. (Eskola ym., 1996; Eriksson ym., 2016.) Valintakriteereinä käytettiin kokemusta, asiantuntemusta ka vastuullisuuden tuntemusta ohjelmistokehityksessä. Tavoitteena oli kerätä syvällistä aineistoa, joka mahdollistaa tutkittavan ilmiön kokonaisvaltaisen ymmärtämisen. Tutkimuksen empiirinen aineisto kerättiin viidestä ohjelmistokehityksen kannalta relevantista organisaatiosta. Haastateltavat anonymisoitiin tunnuksilla H1–H5 ja organisaatiot tunnuksilla O1–O4. Organisaatiot edustivat erikokoisia ja erilaisissa toimintaympäristöissä toimivia yrityksiä.

Tutkimuksessa oli mukana suuri asiantuntija- ja konsultointiorganisaatio, finanssialan palveluorganisaatio, suuri digitaalisten palvelujen asiantuntijaorganisaatio sekä keskisuuri ohjelmistoalan asiantuntijaorganisaatio. Näin haastatteluaineisto kattoi sekä suurten organisaatioiden vakiintuneita toimintamalleja että keskisuurten asiantuntijaorganisaatioiden ohjelmistokehityskäytäntöjä. Organisaatioiden toimialat vaihtelivat teknisestä konsultoinnista ja digitaalisista ratkaisuista finanssialan digitaalisiin palveluihin, data- ja pilvipalveluihin sekä kriittisten tietojärjestelmien kehittämiseen. Haastateltavat edustivat ohjelmistokehityksen ja digitaalisten ratkaisujen kannalta keskeisiä rooleja. Mukana oli projektinhallinnan, tuoteomistajuuden, ohjelmistokehityksen ja teknisen arkkitehtuurin näkökulmia. Haastateltavien valinta perustui tarkoituksenmukaiseen otantaan: heillä oli työnsä kautta kokemusta ohjelmistokehityksestä, ohjelmistoprojektien käytännöistä, teknisestä päätöksenteosta tai digitaalisten palvelujen kehittämisestä. Tämä mahdollisti kestävyyden tarkastelun sekä teknisestä että organisatorisesta näkökulmasta.

Ohjelmistotekniikan empiirisessä tutkimuksessa suositellaan tutkittavan ilmiön tarkastelua useiden roolien ja näkökulmien kautta, jotta voidaan tunnistaa sekä organisatorisia että teknisiä vaikutusmekanismeja (Runeson & Höst, 2009). Tässä tutkimuksessa tätä periaatetta sovellettiin valitsemalla haastateltavia, joiden työ liittyy ohjelmistokehityksen eri vaiheisiin ja päätöksenteon tasoihin.

Haastateltavien osallistuminen kestävyyteen liittyvään päätöksentekoon tai käytäntöihin vaihteli. Osalla osallistuminen oli epäsuoraa, jolloin kestävyys näkyi esimerkiksi projektien, asiakasvaatimusten, teknisten ratkaisujen tai organisaation toimintamallien kautta. Kahdella

haastateltavalla osallistuminen oli suoraa, mikä tarkoittaa, että heillä oli välitöntä kokemusta kestävyyteen liittyvien näkökulmien käsittelystä ohjelmistokehityksen yhteydessä. Tämä vaihtelu on tutkimuksen kannalta hyödyllistä, koska se mahdollistaa sen tarkastelun, näkyykö kestävyys organisaatioissa systemaattisena toimintamallina vai yksittäisten roolien, projektien ja henkilöiden kautta.

Tutkijan aiempi työelämäkokemus ohjelmistoihin, liiketoimintaprosesseihin ja organisaatioiden toimintaympäristöihin liittyvissä tehtävissä tuki tutkimusaiheen ymmärtämistä ja haastatteluteemojen muodostamista. Samalla tutkimuksessa pyrittiin säilyttämään analyyttinen etäisyys aineistoon siten, että tulkinnat perustuivat haastatteluaineistoon ja teoreettiseen viitekehykseen, eivät tutkijan ennakko-oletuksiin.

Haastateltavan tunnus	Organisaatio	Organisaatiotyyppi	Toimiala	Organisaation koko (henkilöä)	Haastateltavan rooli	Kokemus (vuotta)	Osallistuminen kestävyyteen (Suora: S Epäsuora: E)	Haastattelun pituus (min)	Haastattelu (Teams: T, Live: L)	Päivämäärä
H1	O1	Asiantuntija- ja konsultointiorganisaatio	Tekninen konsultointi ja digitaaliset ratkaisut	+ 1000	Projektipäällikkö	5	E	48	T	15.4.2026
H2	O2	Finanssialan palvelu-organisaatio	Finanssipalvelut ja digitaaliset palvelut	+ 600	Tuoteomistaja/ ohjelmistokehittäjä	20	E	57	T	21.4.2026
H3	O3	Digitaalisten palvelujen asiantuntijaorganisaatio	Ohjelmistokehitys, data- ja pilvipalvelut	+ 1000	Ohjelmistokehittäjä	7	S	58	L	22.4.2026
H4	O3	Digitaalisten palvelujen asiantuntijaorganisaatio	Ohjelmistokehitys, data- ja pilvipalvelut	+ 1000	Tekninen arkkitehti	22	S	55	T	30.4.2026
H5	O4	Ohjelmistoalan asiantuntijaorganisaatio	Kriittiset digitaaliset palvelut ja tietojärjestelmät	+ 100	Arkkitehti	25	E	37	T	5.5.2026

Taulukko 2. Haastateltavien ja organisaatioiden kuvaus.

Tutkimusprosessin avoin dokumentointi on keskeinen osa laadullisen tutkimuksen luotettavuuden arviointia (Hirsjärvi ym. 2007). Tässä luvussa kuvataan haastateltavien

valintaperusteet, organisaatiokonteksti, aineistonkeruun toteutus, analyysimenetelmä sekä aineiston anonymisointi ja rajaukset. Koska tutkimuksen aineisto koostuu viidestä asiantuntijahaastattelusta, aineistoa ei käsitellä tilastollisesti edustavana otoksena. Aineiston riittävyyttä arvioidaan laadullisen tutkimuksen periaatteiden mukaisesti sen perusteella, kuinka hyvin se tuottaa tutkimuskysymysten kannalta relevantteja näkökulmia ja mahdollistaa ilmiön monipuolisen tarkastelun.

Kun haastatteluissa kerätty aineisto alkaa toistaa itseään, viittaa se siihen, että se on tavoittanut tutkittavien kysymysten moninaisuuden, syvyyden ja nyanssit (Hennink ym., 2022, s. 2). Kattava aineisto osoittaa tutkimuksen sisältövaliditeetin vahvuuden ja tarkkuuden. Kun tutkimus etenee eikä lisäaineiston sisällyttäminen tuo esiin uusia teemoja, ideoita tai käsitteitä analysoitavaksi, saavutetaan aineiston kyllästyminen (data saturation), josta voidaan todeta, ettei lisäotanta ole enää tarpeen (Farrugia, 2019, s. 69).

3.3 Aineiston keruumenetelmä

Haastatteluaineisto kerättiin puolistrukturoitujen teemahaastattelujen avulla. Puolistrukturoitu haastattelu soveltuu tutkimukseen, koska se mahdollistaa ennalta määriteltyjen teemojen käsittelyn kaikkien haastateltavien kanssa, mutta antaa samalla tilaa haastateltavien omille kokemuksille, esimerkeille ja tulkinnoille. Tämä on perusteltua, koska tutkimuksen kohteena oleva ilmiö on sosio-tekninen ja kontekstisidonnainen.

Haastattelut toteutettiin huhti–toukokuussa 2026. Suurin osa haastatteluista toteutettiin Microsoft Teams -yhteydellä ja yksi haastattelu kasvokkain. Haastattelujen kesto vaihteli noin 48 minuutista 58 minuuttiin. Haastattelut litteroitiin analyysiä varten ja aineisto anonymisoitiin ennen varsinaista analyysiä. Haastatteluissa käsiteltiin kestävyyden merkitystä ohjelmistokehityksessä, kestävyyttä ei-toiminnallisena vaatimuksena, ohjelmistokehityksen elinkaarivaiheita, ohjelmistojen vaikutustasoja, systeemivaikutuksia, mittaamista ja arviointikäytäntöjä, sekä organisaation kypsyyttä, vastuita ja kehitystarpeita. Haastattelurunko muodostettiin tutkimuksen teoreettisen viitekehyksen pohjalta. Kysymykset ryhmiteltiin teemoihin, jotka vastasivat tutkimuksen keskeisiä käsitteellisiä osa-alueita ja toimivat samalla teoriaohjaavan analyysin lähtökohtana. Haastattelukysymykset perustuvat tutkimuksen teoreettisessa viitekehyksessä tunnistettuihin keskeisiin käsitteisiin, teemoihin ja niiden välisiin yhteyksiin. Haastattelukysymykset laadittiin tuottamaan laadullista ymmärrystä siitä, miten kestävyys tunnistetaan, tulkitaan ja huomioidaan ohjelmistokehityksen eri käytännöissä, eikä niiden tarkoituksena ollut mitata valmiiksi rajattuja muuttujia.

3.4 Tulkinta aineiston kattavuudesta

Tutkimuksen aineisto koostuu viidestä puolistrukturoidusta asiantuntijahaastattelusta. Aineiston tarkoituksena ei ole tuottaa tilastollisesti yleistettävää kuvaa ohjelmistoalan tilanteesta, vaan muodostaa laadullinen ja kontekstisidonnainen ymmärrys siitä, miten kestävyys näkyy ohjelmistokehityksen käytännöissä erilaisissa organisaatio- ja roolikonteksteissa.

Haastatteluaineiston kattavuutta vahvistaa se, että haastateltavat edustivat erilaisia näkökulmia ohjelmistokehityksen elinkaareen. Projektinhallinnan näkökulma toi esiin asiakasohjauksen, aikataulujen, kustannusten ja vastuiden merkityksen. Tuoteomistajan ja ohjelmistokehittäjän näkökulma toi esiin vaatimustenhallinnan, priorisoinnin ja toteutuksen välisen yhteyden. Ohjelmistokehittäjän näkökulma auttoi tarkastelemaan, näkyykö kestävyys päivittäisessä kehitystyössä, koodauksessa, optimoinnissa, testauksessa tai teknisissä valinnoissa. Teknisen arkkitehdin näkökulma oli erityisen keskeinen ohjelmistojen elinkaaren, skaalautuvuuden, integraatioiden, datavirtojen, infrastruktuurin ja teknisen kestävyyden kannalta.

Organisaatioiden erilaiset toimialat vahvistavat aineiston kontekstuaalista monipuolisuutta. Finanssialan digitaalisten palvelujen tarkastelu toi esiin sääntelyn, luotettavuuden ja pitkäikäisten järjestelmien näkökulmia. Digitaalisten palvelujen asiantuntijaorganisaatio puolestaan nosti esiin ohjelmistokehityksen, pilvipalvelujen ja asiakasprojektien käytäntöjä. Kriittisiä digitaalisia palveluja kehittävä ohjelmistoalan asiantuntijaorganisaatio täydensi aineistoa järjestelmien toimintavarmuuden, pitkäikäisyyden ja mahdollisten yhteiskunnallisten vaikutusten näkökulmasta.

Vaikka aineisto oli määrällisesti rajallinen, se on tutkimuksen tavoitteeseen nähden tarkoituksenmukainen, koska haastateltavat edustavat ohjelmistokehityksen eri rooleja ja organisaatiokonteksteja. Aineiston avulla voidaan tarkastella, miten kestävyys ymmärretään, missä vaiheissa se huomioidaan ja millaiset tekijät estävät tai tukevat sen systemaattista integrointia ohjelmistokehitykseen.

3.5 Data-analyysi

Laadullisessa tutkimuksessa aineiston analyysillä on keskeinen rooli, sillä sen avulla voidaan tunnistaa tutkittavan ilmiön merkityksiä, suhteita ja rakenteita. Analyysissä aineistosta etsitään yhtäläisyyksiä ja eroja koodaamisen, luokittelun ja tulkinnan avulla. Temaattinen

analyysi soveltuu hyvin tämän tutkimuksen kaltaiseen asetelmaan, jossa tavoitteena on ymmärtää sosio-teknistä ilmiötä useasta näkökulmasta. Menetelmän keskeinen vahvuus on sen joustavuus, minkä ansiosta sitä voidaan soveltaa erilaisiin teoreettisiin viitekehyksiin, tutkimusasetelmiin ja aineistotyyppeihin (Kiger ym., 2020, s. 847).

Tässä diplomityössä analyysimenetelmänä käytetään temaattista analyysiä yhdistettynä laadulliseen sisältöanalyysiin. Sisältöanalyysi mahdollistaa systemaattisen lähestymistavan laadullisen aineiston tarkasteluun ja mahdollistaa sekä eksplisiittisen analyysin, että implisiittisen merkitysten ja rakenteiden tunnistamisen, mikä johtaa kategorioiden ja teemojen muodostumiseen (Lindgren ym., 2020, s. 2). Laadullisen sisältöanalyysin keskeisenä tavoitteena on tuottaa kattava kuvaus ja analyyttinen tulkinta tutkittavasta ilmiöstä (Eriksson ym., 2016, s. 120). Tämä on olennaista, kun tutkimuksessa tarkastellaan ohjelmistokehityksen kestävyyteen liittyviä käytänteitä, vipuvaikutuksia ja organisaation toimintatapoja.

Teema voidaan määritellä yhdistäväksi merkitysrakenteeksi, joka kulkee useiden kategorioiden läpi ja kokoaa yhteen tutkittavan ilmiön keskeisen sisällöllisen ytimen (Lindgren ym., 2020, s. 2). Teemat muodostetaan analyysiprosessin aikana tutkijan systemaattisen tulkinnan kautta siten, että ne vastaavat tutkimuskysymyksiin (Kiger ym., 2020). Tässä tutkimuksessa analyysi toteutettiin teoriaohjaavasti, ja teemojen muodostaminen perustuu yhdistettyyn analyysilogiikkaan, jossa hyödynnetään sekä deduktiivista että induktiivista päättelyä. Deduktiivinen vaihe perustuu tutkimuksen teoreettiseen viitekehykseen, erityisesti kestävän ohjelmistokehityksen ja ohjelmistojen vipuvaikutusten käsitteisiin, joiden perusteella muodostetaan alustava koodausrunko. Induktiivisessa vaiheessa analyysiä täydennetään aineistosta tunnistetuilla uusilla kategorioilla tai teemoilla, joita ei ole ennalta määritelty teoriassa. Tällä tavoin teoria ohjaa aineiston jäsentämistä, mutta ei sulje pois haastatteluissa esiin nousevia uusia havaintoja.

Analyysin tavoitteena on tunnistaa keskeiset mekanismit, jotka voidaan ymmärtää positiivisten kädenjälkivaikutusten aiheuttajiksi. Lisäksi tavoitteena on kuvata näkökulmia, joita voidaan sisällyttää ohjelmistokehityksen eri vaiheisiin kädenjälkivaikutusten tunnistamiseksi ja haitallisten hiilijalanjälkivaikutusten vähentämiseksi. Analyysin perusteella pyritään muodostamaan käytännön toimintamalleja, joiden avulla organisaatiot voivat tunnistaa ohjelmistojen hiilikädenjälkeen liittyviä vipuvaikutuksia. Näiden havaintojen pohjalta rakennetaan poikkitieteellistä viitekehystä kestävyyden huomioimiseksi ohjelmistokehityksessä.

Haastatteluteema	Teoreettinen perusta	Tutkimus-kysymys	Mitä aineistosta etsitään?	Alustava analyysi-luokka
A. Taustatiedot ja ohjelmisto-kehityksen organisointi	Ohjelmistokehityksen elinkaari, SDLC, vesiputousmalli, ketterä kehitys, ohjelmistoprojektin roolit ja päätöksenteko	**TK1,2,3,4**	Haastateltavan roolia, vastuita, päätöksentekovaltaa, osallistumista ohjelmistokehityksen vaiheisiin sekä organisaation kehitysmallia	Organisaatio- ja roolikonteksti
B. Kestävyyden merkitys ohjelmisto-kehityksessä	Kestävä kehitys, kestävä ohjelmistokehitys, yritysten kestävä kehitys ja kestävyyden ulottuvuudet	**TK1**	Miten haastateltavat ymmärtävät kestävyyden ohjelmistokehityksen yhteydessä ja nähdäänkö se strategisena tavoitteena, vastuullisuusteemana, laatuvaatimuksena vai käytännön kehitystyötä ohjaavana tekijänä	Kestävyyden käsitteellistäminen ohjelmisto-kehityksessä
C. Kestävyys ei-toiminnallisena laatuvaatimuksena	Ei-toiminnalliset vaatimukset, ohjelmiston laatuattribuutit, vaatimustenhallinta ja kestävyys vaatimuksena	**TK1**	Tunnistetaanko kestävyys konkreettisena ei-toiminnallisena vaatimuksena vai jääkö se yleiseksi periaatteeksi; miten kestävyysvaatimuksia dokumentoidaan ja operationalisoidaan	Kestävyys ei-toiminnallisena vaatimuksena
D. Kestävyys ohjelmisto-kehityksen eri vaiheissa	Ohjelmistokehityksen elinkaari, vaatimusten määrittely, suunnittelu, arkkitehtuuri, toteutus, testaus, käyttöönotto, ylläpito ja käytöstä poisto	**TK1**	Missä ohjelmistokehityksen vaiheissa kestävyys huomioidaan, missä vaiheessa se konkretisoituu päätöksiksi ja missä se jää vähäiseksi tai näkymättömäksi	Kestävyyden sijoittuminen ohjelmisto-kehityksen elinkaareen
E. Vaikutustasot, kädenjälki ja systeemivaikutukset	ICT:n ensimmäisen, toisen ja kolmannen asteen vaikutukset, Green IT, Green by IT, kädenjälkivaikutukset, rebound-vaikutukset ja systeeminen ajattelu	**TK2**	Tunnistetaanko ohjelmistojen suoria, epäsuoria ja systeemisiä vaikutuksia sekä ohjelmistojen mahdollistamia myönteisiä kädenjälkivaikutuksia ja haitallisia käytönaikaisia vaikutuksia	Suorat, epäsuorat ja systeemiset vaikutukset
F. Mittaaminen, käytännöt ja työkalut	GREENSOFT, Software Carbon Intensity, mittarit, arviointikäytännöt, prosessiarvioinnit, katselmoinnit, Lean-työkalut ja asiakasarvotyökalut	**TK3**	Millaisia mittareita, työkaluja, katselmointeja, prosessikuvauksia tai muita käytäntöjä organisaatioissa käytetään kestävyyden tunnistamiseen, arviointiin ja hallintaan	Mittarit ja arviointi-käytännöt
G. Organisaation kypsyys, esteet ja kehitystarpeet	Kestävä projektinhallinta, yritysten kestävä kehitys, organisaation kypsyys, vastuut, osaaminen, johtaminen ja projektinhallinnan jännitteet	**TK4**	Onko kestävyyden huomioiminen systemaattista vai yksittäisistä henkilöistä riippuvaista; mitkä tekijät estävät kestävyyden integrointia ja mitä tulisi kehittää	Organisaation vastuut, osaaminen, kypsyys, esteet ja kehitystarpeet

Taulukko 3. Haastattelurungon teoriaan sidontataulukko

Teoriasidontamatriisi (taulukko 3) osoittaa, miten haastattelurunko on johdettu tutkimuksen teoreettisesta viitekehyksestä ja miten haastattelukysymykset liittyvät tutkimuskysymyksiin sekä alustaviin analyysiluokkiin. Matriisin tarkoituksena on vahvistaa tutkimusasetelman läpinäkyvyyttä ja osoittaa, että haastattelukysymykset eivät muodosta irrallista kysymyslistaa, vaan ne rakentuvat kestävän ohjelmistokehityksen, vaatimustenhallinnan, ohjelmistokehityksen elinkaaren, ICT:n vaikutustasojen, mittaamisen sekä organisaation kypsyyden teoreettisten kokonaisuuksien varaan.

Teoriaohjaava koodausrunko muodostettiin teoriasidontamatriisin perusteella. Pääkoodit vastaavat tutkimuksen keskeisiä teoreettisia osa-alueita, kuten kestävyyden käsitteellistämistä, kestävyyttä ei-toiminnallisena vaatimuksena, ohjelmistokehityksen elinkaarivaiheita, ICT:n vaikutustasoja, systeemivaikutuksia, mittaamista sekä organisaation kypsyyttä ja kehitystarpeita. Alakoodien avulla aineistosta voidaan tunnistaa tarkempia ilmiöitä ja havaintoja kunkin pääteeman sisällä.

Koodausrunkoa käytetään haastatteluaineiston alustavana jäsentämisen välineenä. Analyysi ei kuitenkaan rajoitu ennalta määriteltyihin koodeihin, vaan aineistosta nousevat uudet havainnot voidaan kirjata erillisiksi induktiivisiksi koodeiksi. Tällä tavoin analyysi säilyttää yhteyden teoreettiseen viitekehykseen mutta mahdollistaa myös aineistolähtöisten teemojen tunnistamisen. Taulukossa 4 esitetään tutkimuksen teoriaohjaava koodausrunko:

Pääkoodi	Alakoodit	Mitä koodilla tunnistetaan aineistosta?	Ensisijainen tutkimuskysymys-yhteys
K1 **Kestävyyden käsitteellistäminen ohjelmistokehityksessä**	K1.1 ympäristöllinen kestävyys K1.2 taloudellinen kestävyys K1.3 sosiaalinen kestävyys K1.4 tekninen kestävyys K1.5 yksilöllinen kestävyys K1.6 strateginen vastuullisuus K1.7 käytännön kehitystyötä ohjaava tekijä	Miten haastateltavat ymmärtävät kestävyyden ohjelmistokehityksen yhteydessä ja mitä kestävyyden ulottuvuuksia he painottavat.	TK1
K2 **Kestävyys ei-toiminnallisena vaatimuksena**	K2.1 kestävyys laatuattribuuttina K2.2 kestävyys NFR-vaatimuksena K2.3 kestävyys yleisenä periaatteena K2.4 kestävyys hyväksymiskriteerinä K2.5 dokumentoitu kestävyysvaatimus K2.6 implisiittinen kestävyysvaatimus K2.7 kestävyysvaatimusten	Tunnistetaanko kestävyys konkreettisena ohjelmistovaatimuksena vai jääkö se yleiseksi tavoitteeksi tai arvoksi.	TK1
K3 **Kestävyys ohjelmistokehityksen elinkaaren vaiheissa**	K3.1 vaatimusmäärittely K3.2 projektin alkuvaihe / front end K3.3 suunnittelu K3.4 arkkitehtuuri K3.5 toteutus K3.6 testaus K3.7 käyttöönotto K3.8 ylläpito ja käyttö K3.9 käytöstä poisto	Missä ohjelmistokehityksen vaiheissa kestävyys tunnistetaan, missä se konkretisoituu päätöksiksi ja missä se jää vähäiseksi.	TK1
K4 **Ohjelmistojen suorat, epäsuorat ja systeemiset vaikutukset**	K4.1 suorat vaikutukset K4.2 energiankulutus K4.3 infrastruktuurin kuormitus K4.4 epäsuorat vaikutukset K4.5 prosessimuutokset K4.6 käyttäytymisen muutos K4.7 kysynnän muutos K4.8 systeemiset vaikutukset K4.9 vaikutusten tunnistamatta jääminen	Tunnistetaanko ohjelmistojen ensimmäisen, toisen ja kolmannen asteen vaikutuksia sekä niiden syntymekanismeja.	TK2
K5 **Kädenjälki, rebound-vaikutukset ja haitalliset käytönaikaiset vaikutukset**	K5.1 positiivinen kädenjälki K5.2 resurssien säästö K5.3 päästöjen vähentäminen K5.4 toiminnan tehostuminen K5.5 rebound-vaikutus K5.6 kulutuksen lisääntyminen K5.7 haitallinen käytönaikainen vaikutus K5.8 virheellisen tiedon aiheuttama vaikutus K5.9 monimutkaisuuden lisääntyminen	Millaisia myönteisiä ja kielteisiä vaikutuksia ohjelmistot voivat aiheuttaa käytön aikana ja laajemmassa toimintaympäristössä.	TK2

K6 **Mittarit, työkalut ja arviointikäytännöt**	K6.1 energiankulutuksen mittaus K6.2 suorituskykymittarit K6.3 päästömittarit K6.4 SCI tai vastaava mittaristo K6.5 laadullinen arviointi K6.6 katselmoinnit K6.7 prosessiarvioinnit K6.8 backlog-kriteerit K6.9 arkkitehtuuriohjaus K6.10 mittareiden puuttuminen K6.11 mittaamisen epäsuoruus tai epävarmuus	Miten kestävyyttä arvioidaan, mitataan, seurataan tai hallitaan ohjelmistokehityksessä.	TK3
K7 **Organisaation vastuut, osaaminen ja kypsyys**	K7.1 vastuunjako K7.2 roolit K7.3 johdon tuki K7.4 osaaminen K7.5 koulutustarpeet K7.6 vastuullisuustavoitteiden yhteys kehitystyöhön K7.7 systemaattinen toimintamalli K7.8 yksittäisistä henkilöistä riippuva toiminta K7.9 organisaation kypsyystaso K7.10 asiakasohjautuva kestävyys K7.11 epävirallinen kestävyysasiantuntijuus / sisäinen kestävyysagentti	Onko kestävyys institutionalisoitunut osaksi organisaation ohjelmistokehitystä vai riippuuko se yksittäisistä henkilöistä, rooleista tai projekteista.	TK3, TK4
K8 **Esteet, ristiriidat ja kehitystarpeet**	K8.1 kiire K8.2 kustannuspaine K8.3 aikataulupaine K8.4 asiakkaan vaatimusten puute K8.5 mittarien puute K8.6 vastuiden epäselvyys K8.7 osaamisvaje K8.8 tekniset esteet K8.9 organisatoriset esteet K8.10 kehitysehdotukset K8.11 standardien tai ohjeiden tarve K8.12 tekoälyn aiheuttama laatu-, kustannus- tai kilpailupaine K8.13 teknologinen abstraktio ja vaikutusten näkymättömyys K8.14 sääntelyn puute ja regulaatio-ohjauksen tarve	Mitkä tekijät estävät kestävyyden huomioimista ja mitä organisaatioissa pitäisi muuttaa kestävyyden paremmaksi integroimiseksi.	TK3, TK4
K9 **Aineistosta nousevat uudet teemat**	K9.1 uusi havainto K9.2 poikkeava näkemys K9.3 uusi käsite K9.4 odottamaton esimerkki K9.5 tutkimuskysymyksiä täydentävä havainto K9.6 digitaalinen suvereniteetti ja resilienssi kestävyyden osana	Sellaiset haastatteluaineistosta nousevat havainnot, joita ei voida sijoittaa suoraan ennalta määriteltyihin teoriaohjaaviin pääkoodeihin.	Tukee kaikkia

Taulukko 4. Teoriaohjaava koodausrunko

Haastatteluaineiston analysoitiin vaiheittain välittömästi jokaisen haastattelun jälkeen. Ensimmäisessä vaiheessa Microsoft Teams tallennetut haastattelut litteroitiin transcription.utu.fi ohjelmalla ja litteroitu teksti korjattiin selkokieliseksi GPT 5.5 Thinking versiolla, jotta litteroitu teksti oli ymmärrettävää. Selkokielisestä tekstistä poistettiin ja anonymisoitiin tunnistettavia kohtia siten, ettei yksittäisiä henkilöitä tai organisaatioita voida tunnistaa jälkeen päin aineistosta. Tämän jälkeen aineisto käytiin kokonaisuutena läpi, jotta muodostettiin yleiskuva haastateltavien näkemyksistä ja aineiston keskeisistä sisällöistä. Jokainen haastattelu käsiteltiin teemoittain tekoälyn avulla, joka mahdollisti kaikki teemakohtaiset löydökset litteroidusta aineistosta analyysiä tehtäessä.

Tämän jälkeen aineisto koodattiin teoriasta nousseiden teemojen perusteella muodostetun koodausrungon avulla. Koodauksessa aineistosta tunnistetiin tekoälyä käyttäen ilmauksia ja esimerkkejä, jotka liittyvät tutkimuksen keskeisiin teemoihin. Haastattelun aineistokatkelma saattoi saada useamman koodin, mikäli se liittyi useaan analyysiluokkaan. Analyysissä nousi esiin myös havaintoja, joita ei voitu suoraan liittää ensimmäisellä kerralla määriteltyihin koodeihin. Näille tehtiin lisäkoodit koodausrunkoon.

Kolmannessa vaiheessa koodatut havainnot ryhmiteltiin pää- ja alateemoihin. Tässä vaiheessa saatua aineistoa tarkasteltiin vastauksista nousevien yhtäläisyyksien, erovaisuuksien ja rinnakkaisten havaintojen valossa.

Lopuksi aineisto analysoitiin ja tulkittiin vertailemalla tutkimuskysymyksiä ja teoreettisesta viitekehystä. Tavoitteena oli tunnistaa, miten kestävyys näkyy ohjelmistokehityksen käytännöissä, missä vaiheissa se huomioidaan tai jää huomioimatta, millaisia käytönaikaisia ja systeemisiä vaikutuksia ohjelmistoihin liitetään sekä millaisia käytäntöjä, mittareita ja kehitystarpeita organisaatioissa tunnistetaan. Tulokset avataan teemoittain luvussa 5.

3.6 Aineiston anonymisointi ja rajaukset

Haastateltavien ja organisaatioiden anonymiteetin turvaamiseksi tutkimuksessa poistettiin henkilöihin ja organisaatioihin liittyvät tunnistetiedot. Haastateltavat merkitään tunnuksilla H1–H5 ja organisaatiot tunnuksilla O1–O4. Tuloksia tarkastellaan rooli-, teema- ja organisaatiotyyppien tasolla, ei yksittäisten yritysten arviointina. Haastatteluista on muutettu sanoja, käytännön työkalujen nimityksiä tai ilmauksia, joiden kautta organisaatiot olisivat tunnistettavissa julkisesti saatavilla olevan tiedon kautta.

Aineiston rajauksena on, että tutkimus perustuu viiteen asiantuntijahaastatteluun ja niiden laadulliseen analyysiin. Tavoitteena ei ole esittää määrällisesti yleistettäviä tuloksia koko ohjelmistoalasta, vaan kuvata, miten kestävyyttä tunnistetaan, tulkitaan ja viedään käytäntöön tutkimukseen osallistuvien asiantuntijoiden ja organisaatioiden näkökulmasta.

Aineistossa painottuvat ohjelmistokehityksen, teknisen suunnittelun, projektityön ja digitaalisten palvelujen näkökulmat. Tutkimus ei arvioi organisaatioiden vastuullisuustyötä kokonaisuutena, vaan tarkastelee kestävyyden näkymistä ohjelmistokehityksen vaatimuksissa, arkkitehtuurissa, toteutuksessa, mittaamisessa, vastuissa ja elinkaaren hallinnassa.

4 Tulokset ja analyysi

Tässä luvussa esitetään tutkimuksen empiirisen aineiston keskeiset tulokset ja analysoidaan niitä tutkimuksen teoreettisen viitekehyksen näkökulmasta. Empiirinen aineisto koostuu viidestä puolistrukturoidusta asiantuntijahaastattelusta, jotka on merkitty tunnuksilla H1–H5. Haastateltavat edustivat ohjelmistokehityksen, projektinhallinnan, tuoteomistajuuden, arkkitehtuurin, kestävän digitalisaation ja organisaation vastuullisuuskäytäntöjen näkökulmia. Tämän vuoksi aineisto mahdollisti kestävyyden tarkastelun sekä teknisenä, organisatorisena että käytännön ohjelmistokehitykseen liittyvänä ilmiönä.

Analyysi perustuu teoriaohjaavaan sisällönanalyysiin. Haastatteluaineistoa tarkasteltiin kestävän ohjelmistokehityksen, ohjelmiston elinkaaren, ei-toiminnallisten vaatimusten, ICT:n vaikutustasojen, kädenjälkivaikutusten, mittaamisen sekä organisaation vastuiden ja kypsyyden näkökulmista. Teoreettinen viitekehys ohjasi aineiston jäsentämistä, mutta analyysissä huomioitiin myös haastatteluista esiin nousseet lisähuomiot, jotka eivät suoraan sopineet ennalta määriteltyihin teemoihin.

Analyysissä tunnistettuja aineistolähtöisiä havaintoja olivat erityisesti tekoälyn käytön vaikutus ohjelmistokehityksen kestävyyteen, teknologisen abstraktion aiheuttama vaikutusten näkymättömyys sekä arkkitehtuuriin ja teknologiavalintoihin liittyvät riippuvuudet. Näitä havaintoja käsiteltiin luvussa empiirisinä havaintoina tai teoriaohjattuina tulkintoina.

Analyysin tavoitteena ei ollut arvioida yksittäisiä haastateltavia tai organisaatioita, vaan muodostaa kokonaiskuva siitä, miten kestävyys ymmärretään ja miten se näkyy ohjelmistokehityksen käytännöissä. Tuloksia tarkasteltiin teema- ja roolitasolla, eikä aineistoa käsitelty tilastollisesti yleistettävänä kuvauksena koko ohjelmistoalasta. Aineiston tarkoituksena oli antaa kuva siitä, millaisia näkemyksiä ja käytännön havaintoja suomalaisissa ohjelmisto-organisaatioissa toimivilla asiantuntijoilla oli kestävyyden huomioimisesta ohjelmistokehityksessä.

Haastatteluissa korostuivat erityisesti vaatimustenhallinta, arkkitehtuurivalinnat, toteutuksen käytännöt, ylläpito, mittaaminen, asiakasohjaus, kustannus- ja aikataulupaineet sekä organisaation vastuut. Osa haastateltavista tarkasteli kestävyyttä ensisijaisesti teknisen pitkäikäisyyden, ylläpidettävyyden ja resurssitehokkuuden näkökulmasta, mutta jotkut toivat esiin myös sosiaalisen, eettisen ja ympäristöllisen kestävyyden näkökulmia. Tämä vaihtelu oli

tutkimuksen kannalta olennainen havainto, koska se osoitti, että kestävyyden käsite ei ole ohjelmistokehityksen käytännöissä vielä yhtenäisesti määritelty.

Tulokset esitetään teemoittain. Ensin tarkastellaan, miten haastateltavat käsitteellistävät kestävyyden ohjelmistokehityksen yhteydessä. Tämän jälkeen analysoidaan, tunnistetaanko kestävyys ei-toiminnallisena vaatimuksena tai ohjelmiston laatuattribuuttina. Seuraavaksi tarkastellaan, missä ohjelmistokehityksen elinkaaren vaiheissa kestävyys tulee aineiston perusteella esiin ja missä vaiheissa se jää vähemmälle huomiolle. Tämän jälkeen analyysi kohdistuu ohjelmistojen suoriin, epäsuoriin ja systeemisiin vaikutuksiin sekä siihen, miten haastateltavat tunnistavat ohjelmistojen myönteisiä kädenjälkivaikutuksia ja mahdollisia haitallisia käytönaikaisia vaikutuksia. Näitä käsitteitä käytetään analyysissä teoriaohjattuina tulkintakäsitteinä, vaikka haastateltavat eivät aina itse käytä samoja termejä.

Luvun loppuosassa tarkastellaan mittareita, työkaluja ja arviointikäytäntöjä sekä organisaation vastuita, osaamista ja kypsyyttä. Lopuksi kootaan yhteen keskeiset esteet, ristiriidat ja kehitystarpeet, jotka aineiston perusteella vaikeuttavat kestävyyden systemaattista integroimista ohjelmistokehityksen käytäntöihin.

4.1 Kestävyyden käsitteellistäminen ohjelmistokehityksessä

Haastatteluissa tuli ilmi, että kestävyys ymmärretään ohjelmistokehityksessä moniulotteisena mutta vielä jäsentymättömänä käsitteenä. Haastateltavat eivät liittäneet kestävyyttä vain ympäristövaikutuksiin, vaan käsitteeseen yhdistyvät tekninen pitkäikäisyys, ylläpidettävyys, taloudellinen tehokkuus, sosiaalinen vastuu, käyttäjävaikutukset, tietoturva, eettiset kysymykset ja organisaation vastuullisuustyö. Tämä näkyi erityisesti siinä, että haastateltavat lähestyivät kestävyyttä omasta roolistaan käsin: arkkitehti painotti elinkaarta ja ajopaikkaa, kehittäjä tehokkuutta ja resurssien käyttöä, projektipäällikkö pitkäikäisyyttä ja asiakkaan tavoitteita sekä vastuullisuustyössä mukana oleva haastateltava sosiaalista ja eettistä ulottuvuutta.

> ”Helposti kestävyys nähdään vihreänä softana, eli että ohjelmisto pyrkii olemaan luonnolle parempi. Itse näen siinä kuitenkin useammankin kulman. Ensimmäinen liittyy siihen, kuinka ympäristöystävällistä koodi on eli kuinka tehokkaasti se on kirjoitettu. Toinen liittyy siihen, missä koodia ajetaan. Ajetaanko sitä esimerkiksi datakeskuksessa, joka pyörii hiilivoimalla, vai ympäristössä, jossa energiankäyttö on kestävämpää? […] Lisäksi kestävyys liittyy siihen, kuinka pitkäikäistä koodi on. Pyöriikö se esimerkiksi viikon ajan mainoskampanjassa vai onko se kymmeniä vuosia käytössä kriittisissä järjestelmissä? Olennaista on siis myös ohjelmiston elinkaari ja se, kuinka usein ohjelmisto pitää kirjoittaa uudelleen.

> Uudelleenkirjoittamisesta syntyy sekä kustannuksia että ympäristövaikutuksia.” (H5)

> ”Kestävyyden voi karkeasti jakaa kahteen asiaan sen pohjalta, mitä olemme tehneet eettisen digitalisaation ja ympäristöystävällisen digitalisaation oppaiden kautta. On sosiaalinen kestävyys ja ympäristökestävyys. Sosiaalinen puoli on huomattavasti kehittyneempi. […] Ajattelen, että sosiaalinen vastuu liittyy siihen, mitä tehdään, kenelle tehdään ja miksi tehdään. Ei tehdä esimerkiksi käyttäjiä huijaavia ratkaisuja tai muita haitallisia asioita. […] Ympäristökestävyys tai ympäristövastuullisuus tarkoittaa puolestaan sitä, että ohjelmistoa tehdessä mietitään, miten sen valmistus kuormittaa ympäristöä, miten sen käyttö kuormittaa ympäristöä ja mikä sen loppu on.” (H3)

Kestävyys ei ole haastatteluaineistossa yksiselitteinen tai yhdenmukaisesti määritelty käsite. H5 jäsensi kestävyyden ohjelmiston tehokkuuden, ajopaikan ja elinkaaren kautta, kun taas H3 erotti toisistaan sosiaalisen kestävyyden ja ympäristökestävyyden. Molemmissa vastauksissa kestävyys kuitenkin ymmärrettiin ohjelmiston koko vaikutusketjuna eikä vain yksittäisenä teknisenä ominaisuutena.

Tekninen kestävyys oli aineistossa yksi helpoimmin tunnistettavista lähtökohdista. Se liitettiin erityisesti ohjelmiston pitkäikäisyyteen, ylläpidettävyyteen, muokattavuuteen, robustisuuteen, suorituskykyyn ja teknisen velan hallintaan. Tämä näkyi siinä, että osa haastateltavista hahmotti kestävyyden ensisijaisesti aikaa kestävänä ratkaisuna, jota voidaan ylläpitää ja kehittää pitkällä aikavälillä.

> ”Jos minulta olisi kysytty kolme vuotta sitten, olisin sanonut, että kestävyys liittyy enemmän talouteen. Silloin ajattelin, että kyse on kestävän ohjelmiston tekemisestä: sellaisesta ohjelmistosta, jota on mahdollista kehittää pitkälle tulevaisuuteen. Ei tehdä vain sellaisia ratkaisuja, jotka tehdään kerran ja joita ei sen jälkeen voida parantaa, muokata tai päivittää. Nykyään sanoisin, että myös tekoälyn tuoma näkökulma on nostanut ympäristöpuolen paljon vahvemmin esiin. Onhan ympäristönäkökulma ollut mukana aiemminkin, esimerkiksi siinä, että tehdään koodia, joka käyttää vähemmän resursseja ja kuormittaa siten vähemmän ympäristöä. Tekoäly on kuitenkin tehnyt tästä näkökulmasta paljon suuremman.” (H1)

> ”Minulle tulee kestävyydestä ensimmäisenä mieleen se, että tehdään aikaa kestäviä ratkaisuja. Toisaalta mieleen tulevat myös perinteisemmät kestävyyden määreet, kuten robustisuus, suorituskyky ja sen tyyppiset asiat. Eli ratkaisu kestää myös käyttöä. Ymmärrän kuitenkin, että tässä haetaan varmaan myös luonnon kannalta kestävyyttä: sitä, kuinka paljon jokin sovellus tai ratkaisu käyttää luonnonvaroja, energiaa, erilaisia raaka-aineita tai muita vastaavia resursseja. Sitä näkökulmaa en ole kyllä urallani oikeastaan koskaan miettinyt.” (H2)

H1:n ja H2:n vastaukset osoittivat, että tekninen ja taloudellinen kestävyys olivat ohjelmistokehityksen käytännöissä helpommin tunnistettavia kuin ympäristöllinen kestävyys.

Kestävyys liitettiin ensin ohjelmiston kykyyn säilyä käyttökelpoisena ja ylläpidettävänä. Ympäristöllinen näkökulma tunnistettiin, mutta se ei ollut osa ohjelmistokehityksen arkista ajattelua. H2:n vastauksesta käy ilmi, että ympäristöllinen kestävyys oli asiantuntijalle käsitteellisesti ymmärrettävä, mutta käytännön työssä uusi näkökulma.

> ”Jos ajatellaan tätä puolta, niin silloin tullaan ennen kaikkea tehokkuuteen, esimerkiksi koodin tehokkuuteen. Miten tehokasta koodi on logiikaltaan ja kuinka paljon se kuluttaa suoraan prosessoritehoa. Siitähän siinä viime kädessä on kysymys. Se on varmaan se kulma, jota tässä pitäisi miettiä.” (H2)

Ympäristöllinen kestävyys konkretisoitui haastatteluissa erityisesti resurssien käytön, energiankulutuksen, pilviympäristöjen, datakeskusten ja tekoälyn kautta, mutta ei näkynyt erillisenä ympäristövaatimuksena, vaan teknisen tehokkuuden tai kustannustehokkuuden sivuvaikutuksena.

> ”Konkreettisesti tämä tarkoittaa esimerkiksi sitä, missä ohjelmistot pyörivät, missä konesaleissa ne sijaitsevat, meneekö hukkaenergiaa ja kuinka paljon sähköä ohjelmisto kuluttaa silloin, kun se pyörii. Lisäksi pitää miettiä, mikä sen loppusijoituspaikka on: jääkö se ikuisiksi ajoiksi bittivaruuteen. Uutena asiana on tullut tekoälyn käyttö, joka kuuluu näihin molempiin näkökulmiin. Mitä on vastuullinen tekoälyn käyttö?” (H3)

Ympäristöllinen kestävyys ohjelmistokehityksessä nähtiin vasta silloin, kun se voitiin kytkeä konkreettisiin teknisiin kysymyksiin. Tällaisia olivat esimerkiksi koodin tehokkuus, prosessoritehon käyttö, palvelimen ajopaikka, datakeskuksen energiaprofiili, hukkakäyttö ja tekoälyn resurssikulutus. Tulokset ovat yhteneväiset teoriaosuudessa esitettyyn, sillä aikaisemman tutkimuksen mukaan kestävyyttä ei voida rajata pelkkään ohjelmiston kehittämisvaiheeseen, vaan se liittyy myös ohjelmiston käyttöön, infrastruktuuriin ja elinkaaren loppuun.

Taloudellinen kestävyys kytkeytyi empiirisessä aineistossa vahvasti tekniseen ja ympäristölliseen kestävyyteen. Kestävyyttä rakentavat ratkaisut perusteltiin käytännössä kustannusten, tehokkuuden tai resurssien säästämisen kautta. Tämä tarkoittaa, että ympäristöllisesti parempi ratkaisu voi tulla esiin vasta silloin, kun se voidaan esittää myös taloudellisesti järkevänä ratkaisuna.

> ”Käytännössä melkein kaikki mainitsemasi asiat menevät tavalla tai toisella resurssitehokkuuden kautta. Esimerkiksi testauksessa voidaan miettiä sitä, että CI-putkessa ajetaan vähemmän testejä jokaiselle uudelle ominaisuudelle tai että testit saadaan tehtyä tehokkaammin. Tämä nopeuttaa sitä, miten kehittäjä pystyy saamaan asioita aikaan, kun ei tarvitse odotella testien ajamista GitHubissa tuntikaupalla. Vastaavasti se, että käytössä on juuri sen verran resursseja kuin

> palvelu tarvitsee ja että palvelu skaalautuu automaattisesti ylös ja alaspäin, liittyy asiakkaan ajokustannuksiin. Tällaiset asiat ovat sellaisia, jotka voidaan tavalla tai toisella johtaa jo tällä hetkellä rahatehokkuuteen.” (H4)

Kestävyyden käytännön edistäminen tapahtui H4:n vastauksen perusteella usein välillisesti. Resurssitehokkuutta, automaattista skaalautumista ja testauksen optimointia ei perusteltu ensisijaisesti ympäristösyillä, vaan ajansäästöllä, kustannuksilla ja tehokkuudella. Vastauksissa nähtiin, että ratkaisut voivat kuitenkin vähentää turhaa laskentaa, hukkakäyttöä ja ylimitoitettua infrastruktuuria. Taloudellinen kestävyys johti ympäristölliseen kestävyyteen.

Sosiaalinen ja eettinen kestävyys näkyivät aineistossa erityisesti käyttäjävaikutusten, saavutettavuuden, yhdenvertaisuuden, datan käsittelyn, tietoturvan ja vastuullisen suunnittelun kautta. Tämä laajensi kestävyyden käsitettä teknisen ja ympäristöllisen näkökulman ulkopuolelle. Ohjelmiston kestävyys ei tarkoittanut vain sitä, että järjestelmä kuluttaa vähemmän resursseja, vaan myös sitä, ettei ohjelmisto tuota haitallisia vaikutuksia käyttäjille tai yhteiskunnalle.

> ”Sosiaalinen puoli on huomattavasti kehittyneempi. Siihen liittyvät esimerkiksi designerien guideline-ohjeistukset, se ettei tehdä dark patterneja, eikä osallistuta sellaisiin projekteihin, joista on enemmän haittaa kuin hyötyä. Meillä on myös sisäinen auditointi. Ajattelen, että sosiaalinen vastuu liittyy siihen, mitä tehdään, kenelle tehdään ja miksi tehdään. Ei tehdä esimerkiksi käyttäjiä huijaavia ratkaisuja tai muita haitallisia asioita. Lisäksi kestävyyteen kuuluu tietyllä tavalla myös saavutettavuus: ohjelmiston pitäisi olla mahdollisimman laajasti käytettävää.” (H3)

> ”Palvelumuotoilun puolella otamme enemmän kantaa yhteisölliseen aspektiin. Palvelun pitäisi olla kestävää myös yhteisön kannalta. Se ei esimerkiksi saa syrjiä mitään väestöryhmiä, jotka käyttävät tiettyä palvelua. Lisäksi meillä on etiikkatarkastelu, jossa minäkin olen mukana. Sen kautta tarkastellaan, että projektit, joita teemme, ovat jotenkin huomioituja tästä näkökulmasta. Emme halua tehdä suoranaisesti haitallisia projekteja.” (H4)

Sosiaalinen ja eettinen kestävyys liittyivät ohjelmiston tarkoitukseen, käyttäjäryhmiin ja vaikutuksiin. H3 näki etiikan liittyvän rehellisyyteen ja läpinäkyvyyteen. H4 liitti kestävyyden syrjimättömyyteen ja projektien eettiseen arviointiin. Molemmissa vastauksissa näkyi pyrkimys välttää haitallisten projektien tekemistä.

Aineistossa nousi esiin myös organisaation yleisen vastuullisuuden ja ohjelmistokehityksen konkreettisen kestävyyden välinen ero. Kestävyys voi näkyä organisaation strategiassa, vastuullisuusraportoinnissa, eettisissä periaatteissa tai sisäisissä ohjeissa, mutta tämä ei

tarkoita, että se olisi automaattisesti osa ohjelmistokehityksen vaatimuksia, mittareita tai projektiprosesseja.

> ”Tällaisessa jaossa meillä ei tällä hetkellä ole vielä varsinaista kestävyysteeman huomiointia. Kestävyys huomioidaan tällä hetkellä enemmän yleisellä tasolla. Meillä on esimerkiksi yleisessä Code of Ethicsissä ja Ethical Design Bookletissa yleisellä tasolla kuvattu sitä, miten kestävyysnäkökulmia sekä ympäristö- ja yhteisönäkökulmia pitäisi huomioida. […] Toiminnassa yleensä nämä periaatteet pätevät myös projekteihin, mutta varsinaisessa ohjelmistokehityksen prosessissa kestävyys ei ole vielä tällä hetkellä mukana.” (H4)

> ”Suoraan sanoen ei. Ei kyllä ole. Se ei ole näkynyt missään strategiapapereissa tai muissa vastaavissa. Tai no, firman tasolla strategiassa kestävyys varmasti on jollain tavalla mukana, voisin kuvitella. […] Kyllähän kestävyys silloin jossakin siellä on mukana, mutta ohjelmistokehityksen puolella ei voi sanoa, että se olisi näkynyt minkäänlaisena ohjaavana tekijänä.” (H2)

Organisaatiot voivat yllä olevien vastausten perusteella olla vastuullisuustyössä eri vaiheissa kuin ohjelmistokehityksen käytännöissä. Yritystasolla voitiin puhua vastuullisuudesta, eettisistä periaatteista tai hiilijalanjäljestä, mutta ohjelmistoprojektien arjessa kestävyys ei näkynyt vaatimuksina, hyväksymiskriteereinä, tiketteinä tai mittareina. Tämä oli keskeinen havainto tutkimuskysymysten kannalta, koska se osoittaa eron strategisen vastuullisuuspuheen ja ohjelmistokehityksen operatiivisen käytännön välillä.

Lyhyesti sanottuna, kestävyys ymmärrettiin haastatteluaineistossa laajasti, mutta sen käsitteellinen asema ohjelmistokehityksen arjessa oli vielä vakiintumaton. Haastateltavat tunnistivat teknisen pitkäikäisyyden, ylläpidettävyyden, resurssitehokkuuden, sosiaalisen vastuun, eettiset vaikutukset ja ympäristökuormituksen, mutta nämä näkökulmat eivät muodostaneet yhtenäistä käsitteistöä tai systemaattista toimintamallia. Kestävyys tuli useammassa vastauksessa esiin hyvän teknisen suunnittelun, kustannustehokkuuden, saavutettavuuden, tietoturvan, eettisten periaatteiden tai organisaation vastuullisuustyön kautta. Vastaukset ovat samassa linjassa teoriakatsauksessa esitetyn aikaisemman tutkimusaineiston kanssa.

4.2 Kestävyys ei-toiminnallisena vaatimuksena

Haastatteluaineistossa kestävyys tunnistettiin ohjelmistokehityksessä mahdolliseksi laatuattribuutiksi ja ei-toiminnalliseksi vaatimukseksi, mutta sen asema oli vielä epäselvä ja projektikohtaisesti vaihteleva. Kestävyys ei useimmissa haastatteluissa näyttäytynyt samalla tavalla vakiintuneena vaatimuksena kuin esimerkiksi tietoturva, käytettävyys, saavutettavuus

tai suorituskyky. Sen sijaan se esiintyi usein yleisenä periaatteena, teknisen laadun osana tai asiakkaan, kilpailutuksen, auditoinnin tai organisaation sisäisen ohjeistuksen kautta syntyneenä mahdollisena vaatimuksena.

> ”Itse sanoisin, että omassa työssäni kestävyys näkyy enemmän laatuattribuuttina. Tiedän kuitenkin, että joissakin projekteissa ja joidenkin projektipäälliköiden työssä se nähdään enemmän myös ei-toiminnallisena vaatimuksena. Tähän vaikuttaa paljon se, mitä asiakas toivoo ja mitä asiakas pystyy vaatimaan. Silloin kestävyys voi olla myös toiminnallinen vaatimus siinä mielessä, että se tulee osaksi projektin vaatimuksia. Omassa työssäni se näkyy kuitenkin enemmän laatuattribuuttiajatteluna.” (H1)

Kestävyys ei sijoitu yllä olevan vastauksen perusteella yksiselitteisesti yhteen vaatimuskategoriaan. Se oli joko laatuattribuutti, ei-toiminnallinen vaatimus tai joissakin tilanteissa jopa projektin toiminnallinen vaatimus, jos asiakas määritteli sen konkreettiseksi toteutettavaksi asiaksi. Olennaista oli kuitenkin se, että kestävyyden asema riippui vahvasti siitä, kuka vaatimuksen esitti ja miten se kirjattiin osaksi projektia.

Aineistossa korostui, että kestävyys muuttui konkreettiseksi vaatimukseksi vasta silloin, kun se dokumentoitiin osaksi projektin käytäntöjä, hyväksymiskriteerejä tai valmiin työn määritelmää. Muuten se jäi helposti yleiseksi tavoitteeksi tai yksittäisten asiantuntijoiden harkinnan varaan.

> ”Kyllä uskon, että jos kestävyys olisi asiakkaan vaatimuksena, se dokumentoitaisiin. En ole itse ollut mukana tällaisessa tilanteessa, mutta jos asiakas edellyttäisi, että kestävyyteen liittyvät asiat pitää dokumentoida tai huomioida tietyllä tavalla, ne tulisivat mukaan Definition of Doneen. Eli ennen kuin työ voidaan merkitä valmiiksi, nämä asiat pitäisi käydä läpi. […] Dokumentoinnissa ei ehkä dokumentoida kestävyyttä sinänsä omana erillisenä asiana. Enemmänkin dokumentoidaan asioita, jotka tukevat kestävyyttä. Esimerkiksi sovelluksesta tai järjestelmästä tehdään hyvät dokumentit: miten sitä kehitetään, miten sitä päivitetään ja miten sitä ylläpidetään. Kun perusohjelmistodokumentaatio on kunnossa, se tukee kestävyyttä.” (H1)

Kestävyys tuli yllä olevassa vastauksessa näkyväksi hyväksymiskriteerinä, Definition of Done -määrittelyssä tai dokumentaatiovaatimuksena, mutta tämä ei ollut vielä tavanomainen käytäntö. Useammin kestävyyteen liittyvät vaatimukset, kuten ylläpidettävyys, päivitettävyys ja dokumentaatio, kirjattiin ilman että niitä nimettiin erikseen kestävyysvaatimuksiksi.

Toinen aineistossa näkyvä taso oli implisiittinen kestävyysvaatimus. Tällöin kestävyyttä ei nimetä erilliseksi NFR-vaatimukseksi, mutta se sisältyi teknisiin ratkaisuihin, pitkäikäisiin teknologiavalintoihin, arkkitehtuurin hallittavuuteen, ylläpidettävyyteen, tietoturvaan,

dokumentaatioon tai resurssitehokkuuteen. Tämä näkyi erityisesti H5:n vastauksessa, jossa kestävyys liittyi teknologioiden elinkaareen ja järjestelmän pitkäaikaiseen ylläpitoon.

> ”Kyllä varmaan. Nykyinen työnantajani keskittyy puolustus- ja turvallisuusalan hankkeisiin, ja siellä kestävyys tulee esiin etenkin sitä kautta, että käytetään ratkaisuja, jotka on jo todettu hyviksi ja pitkäikäisiksi. Kyse ei ehkä ole niinkään ympäristönäkökulmasta, vaan siitä, ettei käytetä kaikista uusimpia kirjastoja tai teknologioita. Sitä kautta kestävyys tulee kyllä huomioiduksi. Vaatimuksena voi olla esimerkiksi se, että ratkaisun täytyy pohjautua teknologioihin, jotka ovat olleet olemassa jo jonkin aikaa ja joiden elinkaari tunnetaan. Tällöin voidaan varautua siihen, että jos ohjelmisto on käytössä pitkään, tiedetään myös, miten sitä ylläpidetään.” (H5)

Kestävyys ei välttämättä tarkoittanut ympäristövaatimusta. Se saattoi tarkoittaa myös teknistä pitkäikäisyyttä, luotettavia teknologiavalintoja ja ylläpidettävyyttä. Tällöin kestävyys toimi ei-toiminnallisen vaatimuksen tavoin, vaikka sitä ei nimetty ympäristökestävyydeksi. Teoriaosuudessa esitetty aikaisempien tutkimusten mukaan kestävä ohjelmistokehitys edellyttää, että kestävyys ymmärretään koko ohjelmiston elinkaaren läpäisevänä laatuattribuuttina, ei vain ohjelmiston energiankulutukseen tai vihreään koodaukseen liittyvänä teknisenä kysymyksenä (Naumann ym., 2011; Penzenstadler ym., 2014; Hilty & Aebischer, 2015)

Kolmas taso liittyi auditoitavuuteen ja hyväksyttävyyteen. Joissakin konteksteissa kestävyys tuli mukaan ulkopuolisen tarkastuksen, hyväksynnän tai järjestelmän käyttökelpoisuuden arvioinnin kautta. Tällöin kyse ei välttämättä ollut yksittäisestä mitattavasta asiasta, vaan siitä, että ohjelmiston elinkaari, päivitysmalli ja ylläpidettävyys voitiin dokumentaation avulla osoittaa olevan kestäviä.

> ”Voi olla. Ehkä ne tulevat enemmän esiin juuri pitkäikäisyyden kautta. Ne voivat olla auditoitavia vaatimuksia. Kun ulkopuolinen taho tarkastaa, onko ohjelmisto hyväksyttävä tiettyyn käyttötarkoitukseen, sieltä voi tulla vaatimuksia esimerkiksi siitä, miten päivitysmalli ja muut vastaavat asiat on huomioitu. Sitä kautta kestävyys varmasti tulee mukaan. Siinä mielessä se on ihan validi kriteeri, jos ei puhuta pelkästään hiilidioksidipäästöistä tai muusta ympäristövaikutuksesta, vaan kestävyydestä laajemmin. Tällöin se tulee kyllä esiin.” (H5)

H5:n kuvaama auditoitavuus laajensi kestävyyden tarkastelua NFR:n näkökulmasta. Kun järjestelmän pitkä käyttöikä, päivitysmalli ja ylläpidettävyys olivat kriittisiä, kestävyydestä tuli osa ohjelmiston kriteereitä. Kestävyyden operationalisointi saattoi tapahtua myös dokumentaation, auditoinnin ja elinkaaren hallinnan kautta, ei pelkästään energiankulutuksen mittaamisena.

Aineiston perusteella asiakkaan rooli oli keskeinen siinä, muuttuuko kestävyys varsinaiseksi vaatimukseksi. Konsultti- ja toimitusprojekteissa kestävyyttä voitiin ehdottaa, mutta lopullinen painoarvo riippui usein siitä, onko asiakas valmis maksamaan sen huomioimisesta.

> ”Se olisi hyvin toivottavaa, ja sitä olemme yrittäneet edistää. […] Olemme yrittäneet puskea sitä, mutta sitä ei nähdä samalla tavalla kriittisenä. Isoin syy on tietenkin se, ettei yksikään asiakas ole valmis maksamaan siitä. Ei oikeastaan millään tavalla. Loppujen lopuksi kaikki johtaa siihen, että tehdään sitä, mistä asiakas on valmis maksamaan. Jos joku asiakas sanoisi, että tässä on kaksi tuhatta euroa ylimääräistä ja käyttäkää vaikka kolme päivää enemmän aikaa siihen, että varmistetaan ratkaisun olevan mahdollisimman ympäristöystävällinen, niin sitten se tapahtuisi heti.” (H3)

> ”Toivoisin kovasti, että kestävyydestä saataisiin laatuvaatimuksia, mutta tällä hetkellä sitä pystyy ajamaan enemmän mutkan kautta. Esimerkiksi se, että resursseille on vähemmän hukkakäyttöä, tarkoittaa yleensä myös sitä, että ne maksavat vähemmän. […] Esimerkiksi hiilijalanjäljen pienentämistä pystyy perustelemaan sillä, että vähennämme ajokustannuksia. Se, että samalla voidaan sanoa palvelun olevan kestävämpi, on plussaa. Mutta asiakkaalle tämä pitää usein perustella viivan alla: miksi tällainen ratkaisu on tehty.” (H4)

Kuten vastauksista käy ilmi, kestävyys ei vielä toimi useimmissa projekteissa päätöksenteon perusteena. Sitä perusteltiin kustannustehokkuuden, resurssien säästön tai kustannusten pienentämisen kautta. Kestävyys oli riippuvainen taloudellisista perusteluista. Toisaalta tekninen ja ympäristöllinen kestävyys voitiin toteuttaa silloin, kun ne pystyttiin liittämään asiakkaalle ymmärrettävään taloudelliseen hyötyyn.

Aineistossa kestävyys rinnastettiin myös tietoturvaan ja saavutettavuuteen, mutta siitä erotetaan yksi puute: kestävyydelle ei ole vielä samanlaista sääntelypohjaa, vakiintunutta testauskäytäntöä tai yhteisesti hyväksyttyjä kriteerejä. H5 vertaa kestävyyttä saavutettavuuteen ja nostaa esiin mahdollisuuden, että sääntely voisi tulevaisuudessa muuttaa kestävyyden asemaa ohjelmistokehityksessä.

> ”Näen tässä jonkinlaisia yhtäläisyyksiä esimerkiksi saavutettavuuteen. Saavutettavuuttakaan ei aikaisemmin kauheasti pyydetty, mutta sitä oli kuitenkin tärkeää edistää. Sittemmin siihen on tullut lainsäädäntöä, joka on vienyt asiaa eteenpäin. En tiedä, tuleeko kestävyysnäkökulma jossakin vaiheessa samalla tavalla vaatimukseksi, mutta tällä hetkellä se ei ole kovin vahvasti esillä.” (H5)

Haastateltava H5 näki, että ohjelmistokehityksen laatuvaatimukset voivat muuttua ajan myötä. Saavutettavuus ja tietoturva ovat nykyisin monissa projekteissa selkeitä vaatimuksia, koska niihin liittyy sääntelyä, asiakasodotuksia, testausmenetelmiä ja hyväksymiskriteerejä. Kestävyys vaatimuksena on vasta tämän kehityskaaren alkuvaiheessa.

> ”Se ei ole näkynyt missään strategiapapereissa tai muissa vastaavissa. […] Kyllähän kestävyys silloin jossakin siellä on mukana, mutta ohjelmistokehityksen puolella ei voi sanoa, että se olisi näkynyt minkäänlaisena ohjaavana tekijänä.” (H2)

H2 puolestaa ei nähnyt kestävyyttä lainkaan ohjelmistokehityksen ohjaavana tekijänä, vaikka organisaatiotasolla vastuullisuus ja hiilijalanjäljen laskenta olivat mukana. Organisaatiotason vastuullisuus ei automaattisesti muutu ohjelmistokehityksen ei-toiminnalliseksi vaatimukseksi.

Kokonaisuutena kestävyys tunnistettiin haastatteluaineistossa ohjelmiston mahdolliseksi laatuattribuutiksi ja ei-toiminnalliseksi vaatimukseksi, mutta sen käytännön asema on vielä vakiintumaton. Se saattoi esiintyä eksplisiittisenä vaatimuksena silloin, kun asiakas, kilpailutus, auditointi, sääntely tai organisaation sisäinen prosessi sitä edellyttivät. Useimmin se näkyi kuitenkin implisiittisesti teknisen laadun, ylläpidettävyyden, dokumentaation, pitkäikäisten teknologiavalintojen, resurssitehokkuuden ja kustannusten kautta. Kestävyyden vahvempi asema ohjelmistokehityksessä nähtiin edellyttävän selkeämpää kirjaamista, backlogille, hyväksymiskriteereihin, Definition of Done -määrittelyihin, arkkitehtuuriperiaatteisiin ja mittareihin. Muuten se nähtiin jäävän helposti yleiseksi vastuullisuustavoitteeksi tai yksittäisten asiantuntijoiden kiinnostuksen varaan.

4.3 Kestävyyden huomioiminen ohjelmistokehityksen elinkaaren vaiheissa

Ohjelmiston kestävyys kytkeytyi koko elinkaaren aikaiseen tarkasteluun. Kestävyysvaikutukset muodostuivat vaatimusten, arkkitehtuurivalintojen, toteutuksen, testauksen, käyttöönoton, käytön, ylläpidon ja elinkaaren päättymisen yhteisvaikutuksena. Tämä vastasi GREENSOFT-mallin mukaista cradle-to-grave-ajattelua. Haastatteluaineiston perusteella elinkaariajattelu tunnistettiin, mutta sitä ei vielä sovellettu systemaattisesti kaikissa ohjelmistokehityksen vaiheissa.

Kestävyys tunnistettiin vahvimmin ohjelmistokehityksen alkuvaiheissa eli määrittelyssä, suunnittelussa ja arkkitehtuurissa. Näissä vaiheissa tehtiin ratkaisuja, jotka vaikuttivat ohjelmiston pitkäikäisyyteen, ylläpidettävyyteen, siirrettävyyteen, resurssien käyttöön ja infrastruktuurivalintoihin.

> ”Kyllä se näkyy eniten suunnittelu- ja määrittelyvaiheessa. Kun suunnitellaan arkkitehtuuria, nykyään ei voida suositella kenellekään esimerkiksi konesaliratkaisua. Sen sijaan pyritään tekemään helposti siirrettäviä, päivitettäviä ja muokattavia ohjelmistoja. Se lähtee suunnitteluvaiheesta: tehdään sellainen

> arkkitehtuuri, joka tukee kestävyyttä. Tämä näkyy myös siinä, että suositaan esimerkiksi serverless-ratkaisuja perinteisten serveriratkaisujen sijasta.” (H1)

> ”Olen itse ollut sellaisissa suunnittelutilanteissa tai tarjouksen tekemisen vaiheissa, joissa huomioidaan se, että jonkin voi tehdä nopeasti tai sen voi tehdä hyvin. Siinä tulee esiin juuri tämä näkökulma: jokin ratkaisu kannattaa rakentaa tietyllä tavalla, koska se takaa pitkäkestoisuuden ja sen, että ratkaisu toimii paremmin jatkossa. Usein joutuu kuitenkin tasapainoilemaan sen kanssa, että on kiire, vähän rahaa ja asiakas pohtii, voisiko tämän tehdä jollakin toisella tavalla. […] Kyllä kestävyyttä kuitenkin pyritään huomioimaan suunnitteluvaiheessa. Tarkoituksena ei ole tehdä vain helpoimman kautta, vaan miettiä, miltä ratkaisu näyttää viiden vuoden päästä ja kuka sitä joutuu ylläpitämään.” (H5)

Ohjelmiston elinkaaren alkupäässä tehtiin kestävyyden kannalta tärkeät ratkaisut. Samalla asiakkaan tavoitteet, kilpailutuksen rajaukset sekä aika- ja kustannuspaineet vaikuttivat siihen, kuinka paljon kestävyyteen voidaan käytännössä panostaa. H5:n vastauksessa tuotteen laatu ja pitkäikäisyys, sekä ylläpito pitkällä aikavälillä otettiin huomioon suunnitteluvaiheissa.

Alkuvaiheen merkitys näkyy myös siinä, pääseekö toimittaja tai kehitystiimi vaikuttamaan ratkaisun suunnitteluun. Jos keskeiset vaatimukset ja teknologiat on määritelty jo etukäteen, kestävyyteen liittyvät vaikutusmahdollisuudet kaventuvat.

> ”Kyllä sanoisin, että näitä vaikutuksia mietitään. Omassa roolissani tämä on kuitenkin enemmän asiakkaan vastuulla. Asiakas esimerkiksi miettii, millainen sovelluksen pitäisi olla, ja heillä voi olla tarkat kriteerit sovellukselle. He ovat siis jo suunnitelleet sitä etukäteen. […] Tämä riippuu kuitenkin todella paljon siitä, kuinka tiukka kilpailutus on. Joskus on jo määritelty tarkasti esimerkiksi, mitä pilvipalvelua saa käyttää. Toisessa tilanteessa asiakas voi sanoa vain, että he tarvitsisivat jonkin sovelluksen hoitamaan tietyn asian. Tämä vaikuttaa siihen, millainen meidän roolimme on arvioinnin aikana.” (H1)

Arkkitehtuurivaiheessa kestävyys konkretisoituu erityisesti teknologia-, infrastruktuuri-, pilvi- ja kapasiteettipäätöksissä. Tärkeitä kysymyksiä ovat esimerkiksi kapasiteetin oikea mitoitus, automaattinen skaalautuminen, palveluiden maantieteellinen sijoittelu ja teknologioiden ylläpidettävyys.

> ”Ainakin kapasiteetin mitoittaminen on yksi selkeä asia. Ei tehdä tarpeeseen nähden ylimitoitettua ratkaisua. Pilvipalvelut ovat siinä mielessä hyviä, että niissä autoscalingin saa käyttöön aika näppärästi. Mutta se, osataanko sitä konfiguroida oikein, on toinen kysymys. Sitä ei aina mietitä kovin tarkasti. Se voi olla vain rivinä jossakin, että hyödynnämme autoscalingia. Mutta kuka sen tekee, tehtiinkö se oikeasti ja seurataanko sitä, miten se käyttäytyy? Se on kokonaisuuden miettimistä. Sitten on myös maantieteellinen sijoittelu. Voi olla vaatimuksia, että palvelut pitää sijoittaa esimerkiksi EU-alueelle. Lisäksi tarvitaan ymmärrystä esimerkiksi siitä, että jos Suomeen tulee uusi Microsoftin datakeskus Espooseen, niin osataanko jo suunnitteluvaiheessa sanoa, että laitetaan palvelu pyörimään

> sinne, koska se on ympäristön kannalta parempi paikka kuin jokin muu maa, jossa ei ole esimerkiksi kaukolämpöaspektia huomioitu.” (H5)

H5:n vastauksessa infrastruktuurin kestävyys laajensi elinkaareen liittyvää näkökulmaa ulospäin ympäristökestävyyteen. Toteutus- ja testausvaiheessa kestävyys näkyi useimmiten resurssitehokkuutena, suorituskykynä, skaalautuvuutena ja kustannusten hallintana. Näitä ei aina nimetty kestävyystoimenpiteiksi, vaikka niillä olisi vaikutusta kestävyyteen välillisesti.

> ”Käytännössä toi menee […] niin kuin resurssitehokkuuden kautta jostakin kautta. Esimerkiksi sanotaan, että testauksessa se, että vaikkapa CI-putkessa ajetaan vähemmän testejä jokaiselle uudelle ominaisuudelle, mitä tehdään, tai että ne testit saadaan tehtyä tehokkaammin, niin se nopeuttaa sitä, että miten kehittäjä pystyy saamaan asioita aikaan, kun ei tarvitse odotella testien ajautumista jossain GitHubissa tuntikaupalla. Ja vastaavasti sitten se, että meillä on käytössä sen verran juuri resursseja, mitä palvelu tarvitsee ja se skaalautuu automaattisesti ylös ja alaspäin, niin tämä on sitten taas sen asiakkaan ajokustannusten puolelta.” (H4)

Testauksessa kestävyys tuli esiin lähinnä suorituskyvyn, SLA-rajojen, pilvipalveluiden, mittareiden ja asiakkaan erikseen tilaamien testien kautta. Varsinainen ympäristökestävyyden testaus ei aineiston perusteella ollut vielä vakiintunut käytäntö.

Käyttöönotossa kestävyys liittyi käyttäjien toimintatapojen muutokseen, oppimiseen ja mahdolliseen vastarintaan. Kestävyys ei ollut vain tekninen kysymys, vaan siihen liittyi myös sosiaalinen ja organisatorinen muutos.

> ”Siirtymävaiheessa on kuitenkin aina jonkinlainen pieni negatiivinen vaikutus asiakkaan systeemeihin. Kaikki henkilöt joutuvat opettelemaan uuden järjestelmän käyttöä, ja siinä syntyy usein vastarintaa uuden käyttöönottoa ja tekemistä kohtaan. Silloin uudesta sovelluksesta löydetään helposti kaikki mahdolliset negatiiviset asiat. Kun käyttäjien kanssa keskustellaan, edetään yhdessä ja näytetään, miten ratkaisu auttaa tulevaisuudessa, lopputulos on yleensä hyvä ja positiivinen. […] Enemmän kyse on ollut väliaikaisista negatiivisista vaikutuksista käyttöönoton yhteydessä.” (H1)

Ylläpidossa ja käytöstä poistossa kestävyys konkretisoitui pitkäikäisyyden, teknisen velan hallinnan, päivityskäytäntöjen, vanhojen riippuvuuksien hallinnan ja tarpeettomien ominaisuuksien poistamisen kautta. Aineiston perusteella ylläpito saattoi jäädä minimitasolle, jos asiakas ei resursoinut aktiivista kehittämistä.

> ”Kyllä, sovellusten pitkäikäisyys huomioidaan. […] Meillä on esimerkiksi Gitin tietoturvaskannaukset kaikille projekteille ja muita vastaavia käytäntöjä. Niillä pakotetaan osaltaan sitä, ettei projekteissa olisi kovin vanhoja paketteja tai kirjastoja. Tämä taas helpottaa sovelluksen pitkäikäisyyttä ja kestävyyttä sen elinkaaren aikana tai elinkaaren loppuun asti. Tämä kuitenkin vaihtelee todella paljon asiakkaasta riippuen.” (H1)

> ”Ylläpidossa kyse on jatkuvasta kehittämisestä ja parantamisesta. Ohjelmiston elinkaaren aikana voidaan tehdä muutoksia ja teknisiä parannuksia, jotka tehostavat ohjelmiston toimintaa. Lisäksi voidaan poistaa käytöstä sellaisia ominaisuuksia, joita ei enää tarvita. Sekin voi vaikuttaa kestävyyteen, ettei legacy-koodia jää turhaan pyörimään nurkkiin, jos sitä ei enää tarvita. […] Käytöstä poisto on myös hyvä esimerkki. Siinä tulee usein ongelmia, koska jokin vanha järjestelmä voidaan jättää pyörimään pitkäksi aikaa sillä ajatuksella, että ehkä joku vielä joskus tarvitsee sitä. Pitäisi olla enemmän rohkeutta tehdä päätös, että tämä järjestelmä ajetaan nyt alas, uusi tuli tilalle ja asia on sillä selvä. Se ei ole pelkästään kestävyysasia, vaan helpottaa muutenkin hallinnollista taakkaa.” (H5)

Aineistossa näkyi myös pyrkimyksiä tuoda elinkaarinäkökulma osaksi käytännön prosesseja. Tämä oli kuitenkin vielä keskeneräistä.

> ”Nämä elinkaariasiat on huomioitu siinä meidän checklistissä, joka meidän olisi tarkoitus saada käyttöön. Siinä otetaan huomioon myös käytöstä poistot ja muut vastaavat asiat. Tällä hetkellä näitä ei siis vielä huomioida systemaattisesti, mutta meillä on kova pyrkimys päästä siihen, että ne huomioitaisiin. […] Meillä on ihan konkreettista tekemistä sen eteen, että näin päästäisiin toimimaan, mutta sitä ei ole vielä saatu osaksi prosessia. Tätä checklistiä pilotoidaan […] mutta sitä ei ole vielä saatu laajemmin jalkautettua projekteihin.” (H4)

Kestävyys huomioitiin vahvimmin ohjelmistokehityksen alkuvaiheissa, erityisesti määrittelyssä, suunnittelussa ja arkkitehtuurissa. Toteutuksessa ja testauksessa kestävyys näkyi lähinnä resurssitehokkuutena, suorituskykynä, skaalautuvuutena ja kustannusten hallintana. Käyttöönotossa korostuivat käyttäjien toimintatapojen muutos ja siirtymävaiheen vaikutukset. Ylläpidossa ja käytöstä poistossa kestävyys liittyi teknisen velan hallintaan, aktiiviseen kehittämiseen, turhien ominaisuuksien poistamiseen ja vanhojen järjestelmien hallittuun alasajoon. Kokonaisuutena empiirisestä aineistosta näkyi, että elinkaariajattelu tunnistettiin, mutta sitä ei vielä sovellettu systemaattisesti kaikkiin ohjelmistokehityksen vaiheisiin.

4.4 Ohjelmistojen suorat, epäsuorat ja systeemiset vaikutukset

Ohjelmistojen kestävyyttä voidaan tarkastella suorien, epäsuorien ja systeemisten vaikutusten kautta. Tässä tutkimuksessa jaottelu perustui ICT:n ensimmäisen, toisen ja kolmannen asteen vaikutuksiin (Hilty & Aebischer, 2015). Haastateltavat eivät käyttäneet näitä käsitteitä sellaisenaan, mutta aineistosta voitiin tunnistaa näitä vastaavia havaintoja.

Suorat vaikutukset liittyivät ohjelmiston ja sen infrastruktuurin energiankulutukseen, resurssien käyttöön, ajonaikaiseen kuormitukseen ja teknologiavalintoihin. Ne näkyivät

esimerkiksi koodin tehokkuudessa, pilviympäristön mitoituksessa, datakeskuksen sijainnissa ja automaattisessa skaalautumisessa.

> ”Helposti kestävyys nähdään vihreänä softana, eli että ohjelmisto pyrkii olemaan luonnolle parempi. Itse näen siinä kuitenkin useammankin kulman. Ensimmäinen liittyy siihen, kuinka ympäristöystävällistä koodi on eli kuinka tehokkaasti se on kirjoitettu. Toinen liittyy siihen, missä koodia ajetaan. Ajetaanko sitä esimerkiksi datakeskuksessa, joka pyörii hiilivoimalla, vai ympäristössä, jossa energiankäyttö on kestävämpää? Esimerkiksi Suomeen ollaan rakentamassa Microsoftin palvelinkeskusta, joka tuottaa samalla kaukolämpöä. Siinä mielessä ratkaisu voi olla ympäristöystävällisempi kuin se, että samaa koodia ajettaisiin likaisen energian alueella.” (H5)

> ”Konkreettisesti tämä tarkoittaa esimerkiksi sitä, missä ohjelmistot pyörivät, missä konesaleissa ne sijaitsevat, meneekö hukkaenergiaa ja kuinka paljon sähköä ohjelmisto kuluttaa silloin, kun se pyörii. Lisäksi pitää miettiä, mikä sen loppusijoituspaikka on: jääkö se ikuisiksi ajoiksi bittivaruuteen.” (H3)

Suorat vaikutukset eivät rajautuneet pelkkään koodiin, vaan ne muodostuivat ohjelmiston teknisestä toteutuksesta, infrastruktuurista ja käyttöympäristöstä yhdessä. Suorien vaikutusten tunnistamista vaikeutti teknologinen abstraktio. Kehittäjä ei välttämättä nähnyt, miten koodi, sovelluskehykset, pilvipalvelut ja valmisjärjestelmät vaikuttivat prosessorikuormaan, energiankulutukseen tai infrastruktuurin käyttöön.

> ”Ensimmäisenä tuosta tulee mieleen se, mitä tietotekniikan kehityksessä on tapahtunut: tekninen stack on kasvanut aivan järjettömän korkeaksi. Tarkoitan sitä, että erilaisia kerroksia ja tasoja on nykyään tietotekniikassa valtava määrä. […] Eli stack on niin paksu, että siitä on vaikea hahmottaa enää juuri mitään. Siellä on jo itsessään monta black boxia välissä. Sen lisäksi käytössä voi olla valmis järjestelmä, joka on myös käyttäjälle itselleen black box. Tämä on yksi tekijä. Prosessori on nykyään niin kaukana koodaajasta, ettei ensimmäisenä tule mieleen miettiä, mitä prosessori tekee tai ei tee missäkin kohdassa.” (H2)

Epäsuorat vaikutukset näkyivät erityisesti prosessien, työnkulkujen ja käyttäjien toiminnan muutoksina. Ohjelmisto vähensi manuaalista työtä, siirsi asiointia itsepalveluun, automatisoi työvaiheita tai yhdisti erillisiä järjestelmiä.

> ”Jos ajatellaan esimerkiksi ulkoisille asiakkaille suunnatun verkkopalvelun kehitystä, niin sehän pyrkii nimenomaan tällaisiin systeemisiin vaikutuksiin. Tavoitteena on, että asiakkaiden käyttäytyminen muuttuu siihen suuntaan, etteivät he enää soita meille tai lähetä sähköpostia, vaan menevät verkkopalveluun ja hoitavat asian siellä itse.” (H2)

> ”Kun jokin prosessin osa automatisoidaan, työntekijän ei tarvitse enää tehdä sitä. Hän voi tehdä jotakin muuta. Kyllähän se totta kai vaikuttaa, ja pyrkimyshän on nimenomaan siihen, että se muuttaa toimintaa. […] Voi siis sanoa, että näitä vaikutuksia tunnistetaan. Varmasti tunnistetaan vaikutuksia eri tasoilla.” (H2)

Myös kielteisiä epäsuoria vaikutuksia tuli esille, jos uusi järjestelmä lisäsi käyttäjän työtä tai teki prosessista aiempaa monimutkaisemman.

> ”Tyypillisimpiä tapauksia ovat sellaiset, joissa jonkin tehtävän tekemiseen tarvitaan esimerkiksi kymmenen klikkausta, ja sitten se saadaan suoraviivaistettua sovelluksessa niin, että siihen tarvitaankin enää viisi klikkausta. Toinen hyvin tyypillinen tapaus liittyy kaikenlaisiin Excel- ja copy-paste-prosesseihin. Kun onnistutaan tekemään esimerkiksi integraatio, jolla vältetään Excelin käyttö ja copy-paste, siitä tulee yleensä hyvää palautetta. Silloin voidaan todeta, että saatiin tämäkin lopulta pois työpöydältä.” (H2)

H2:n vastaus kuvasi epäsuoraa vaikutusta onnistumisen kautta eli mitä tyypillisesti integraatioilla tavoiteltiin.

> ”Samalla voidaan kuitenkin sanoa, että vaikutus voi joskus olla myös negatiivinen. Esimerkiksi jos järjestelmä vaihdetaan toiseen, uudessa järjestelmässä voi olla jokin asia, jota ei pystykään tekemään yhtä tehokkaasti kuin edellisessä. Joskus käy niinkin, että klikkausten määrä lisääntyy jossain kohdassa. Tai jos tulee uusi tuote, joka on aluksi toteutettu työkaluihin vähän vaillinaisesti, klikkauksia ja copy-pastea voidaan tarvita enemmän. Se voi olla negatiivinen vaikutus.” (H2)

Systeemiset vaikutukset liittyivät laajempiin muutoksiin käyttäytymisessä, palvelurakenteissa, organisaatioiden toimintatavoissa ja yhteiskunnallisissa käytännöissä. Niitä voitiin lähestyä esimerkiksi käyttäjäpolkujen, käytönaikaisen seurannan ja oletusvalintojen kautta.

> ”.. käytettävyys on tässä hyvä näkökulma. Voidaan esimerkiksi tarkastella, kuinka kauan käyttäjällä menee tietyn toimenpiteen suorittamiseen. Meneekö se yhdellä klikkauksella vai pitääkö käyttäjän navigoida kymmenen eri valikon kautta päästäkseen tekemään jonkin asian? Tämä on jo aika hyvä kestävyysmittari, koska se säästää paljon resursseja. […] Kun nähdään, miten käyttäjät toimivat järjestelmässä, voidaan muokata käyttäjäpolkuja ja optimoida niitä. […] Kyse ei ole vain siitä, että järjestelmä laitetaan tuotantoon ja se on valmis. Sen sijaan pitäisi seurata, miten sitä käytetään, voidaanko sitä parantaa ja voidaanko joitakin osa-alueita jättää kokonaan pois.” (H5)

Systeeminen vaikutus saattoi näkyä myös siinä, että ohjelmisto muutti työn tekemisen paikkaa ja tapaa.

> ” Todella monella asiakkaallamme on ollut esimerkiksi tilanne, jossa heidän olisi pitänyt olla tietokoneen äärellä, vaikka he työskentelevät luonnossa tai muuten kentällä. Koska olemme erikoistuneet paikkatietoon, olemme voineet tehdä samasta ratkaisusta mobiiliversion ja lisätä siihen sijaintitiedon. Tällä on ollut erittäin merkittävä positiivinen vaikutus asiakkaiden tekemiseen ja tällaiseen IT:n kädenjälkeen: siihen, miten he pystyvät tekemään asioita tehokkaammin.” (H1)

Vaikutusten tunnistamista helpotettiin organisaation prosesseilla, kuten checklist-käytännöillä ja eettisellä arvioinnilla. Nämä käytännöt auttoivat tunnistamaan sekä ympäristövaikutuksia että sosiaalisia sivuvaikutuksia, mutta niiden jalkautuminen ohjelmistokehityksen arkeen oli vielä kesken.

> ”Tämän kysymyksen voi ymmärtää myös toisesta näkökulmasta: meillä huomioidaan eri scope-tasojen vaikutuksia. Esimerkiksi firman sisäisessä hiiliraportoinnissa huomioidaan päästölähteitä Scope 3:een asti. Siinä huomioidaan siis myös muiden meidän puolestamme tekemiä päästöjä ja muuta vastaavaa. Toisaalta kysymys voidaan ymmärtää myös sosiaalisten sivuvaikutusten kautta. Erilaisilla ratkaisuilla voi olla isompia ja vähemmän arvattavia vaikutuksia. (H4)

Haastatteluaineistosta tuli esiin suoran, epäsuoran ja systeemisen vaikutustason erottelua, vaikka haastateltavat eivät käyttäneet näitä käsitteitä. Suorat vaikutukset liittyivät koodiin, energiankulutukseen, pilviresursseihin ja infrastruktuuriin. Epäsuorat vaikutukset näkyivät prosessien muutoksina, automaationa, itsepalveluna ja manuaalisen työn vähenemisenä. Systeemiset vaikutukset liittyivät laajempiin toimintatapojen muutoksiin, kuten sähköisen viestinnän oletusvalintoihin, kenttätyön muuttumiseen mobiiliratkaisujen avulla ja käyttäjäpolkujen optimointiin. Keskeistä oli, että vaikutusten tunnistaminen edellyttää sekä teknistä mittaamista että laadullista ymmärrystä siitä, miten ohjelmisto muuttaa käyttäjien, organisaatioiden ja yhteiskunnan toimintaa.

4.5 Kädenjälki, rebound-vaikutukset ja haitalliset käytönaikaiset vaikutukset

Ohjelmistojen kestävyyden arviointi edellyttää suoraa energiankulutusta ja teknistä tehokkuutta laajempaa tarkastelua. Tämä alaluku tarkastelee positiivisen kädenjäljen, haitallisten käytönaikaisten vaikutusten ja rebound-tyyppisten riskien näkökulmaa. Tässä positiivisella kädenjäljellä tarkoitetaan resurssien säästämistä, prosessien tehostumista, työn sujuvoittamista tai käyttäjän toiminnan parantumista. Samalla ohjelmisto voi myös lisätä kuormitusta, kasvattaa käyttöä, siirtää työtä käyttäjälle tai tuottaa haitallisia sosiaalisia ja teknisiä seurauksia.

Haastatteluaineiston perusteella positiivinen kädenjälki ei syntynyt pelkästä digitalisoinnista, vaan siitä, että ohjelmisto todella vähensi hukkaa, paransi työn sujuvuutta tai mahdollisti kestävämmän toimintatavan.

> ”Samalla vaivalla voi tuottaa kestävämpääkin, jos miettii asioita vähän uudella tavalla. Molemmilla osapuolilla on vähän vastuuta. Kestävyydestä ei kuitenkaan

> oikein pysty pyytämään lisähintaa. Sitä on ollut vaikea perustella, jos asiakas ei ole ollut valmiiksi valveutunut. Molempien vastuulla olisi jossain määrin edistää tätä.” (H5)

Kädenjälki syntyi suunnittelun ja toteutuksen tavasta. Jos ohjelmisto tehtiin alusta alkaen pitkäikäisesti ja turhaa kuormitusta välttäen, sama kehitystyö saattoi tuottaa kestävämmän lopputuloksen. Haitalliset käytönaikaiset vaikutukset liittyivät erityisesti siihen, että ohjelmisto jää pyörimään, vaikka sen käyttö väheni, tai että järjestelmään jäi vanhaa koodia ja toiminnallisuuksia, joita ei enää tarvittu.

> ”Yksi asia, joka itse asiassa tulee mieleen, on tekninen velka sovelluksen elinkaaren aikana. Sovelluksessa saattaa olla jostain syystä paljon turhaa koodia. Sovellus, käyttötapa tai jokin muu asia on voinut muuttua niin paljon, että on tehty uusia ratkaisuja, mutta vanha koodi pyörii siellä edelleen. Tavallaan, näin minä sen ymmärrän, se on kestävyyttä vastaan. Joka kerta kun sovellusta käytetään, vanha koodi pyörii siellä mukana. Vanhentunutta tai deprekoitunutta koodia käännetään koko ajan, eikä se ole kovin tehokasta.” (H2)

> ”Toinen asia liittyy juuri tähän vanhentumiseen. Kun ajatellaan sovelluksen elinkaarta, käyttö usein vähenee jossain vaiheessa. Silti sovellus saattaa pyöriä jossain koko ajan ja kuluttaa resursseja. Vähintään se varaa muistia pyöriessään, ja sitä kautta se kuluttaa resursseja turhaan.” (H2)

Yllä olevissa vastauksissa tuli esille haitallinen vaikutus, vaikka ohjelmisto olisi toimnut teknisesti. Ongelma oli siinä, että ohjelmisto jatkoi resurssien kuluttamista ilman riittävää hyötyä. Tämä korosti ylläpidon, käytön vähenemisen ja käytöstä poiston merkitystä kestävyyden arvioinnissa.

Rebound-vaikutukset eivät nousseet haastatteluissa vakiintuneena käsitteenä, mutta aineistosta voitiin tunnistaa rebound-tyyppisiä riskejä. Tällainen riski syntyi esimerkiksi silloin, jos ohjelmiston käyttö helpottui niin paljon, että kokonaiskäyttö kasvoi, tai jos käyttäjä joutui palaamaan palveluun toistuvasti.

> ”Mutta jos käyttäjä käyttää jotakin palvelua normaalikäytössä enemmän, se on yleensä nähty positiivisena asiana. Silloin käyttäjä saadaan palaamaan palveluun uudestaan. Käänteinen näkökulma on kuitenkin se, tarvitseeko käyttäjän ylipäätään olla palvelussa uudestaan. Olisiko parempi, että käyttäjä ei koskaan palaisikaan palveluun, jos hän on kerran käynyt tekemässä tarvitsemansa toimenpiteen? Sekin voi olla toivottu vaihtoehto. Se riippuu palvelusta ja sen tarkoituksesta. Jos käyttäjä käy palvelussa monta kertaa, se voi olla myös negatiivinen asia, jos syynä on se, ettei hän saanut asiaansa kerralla hoidettua loppuun. Tässä on siis kaksi puolta.” (H5)

Suuri käyttömäärä ei ollut kestävyyden näkökulmasta automaattisesti myönteinen asia. Kädenjälkeä piti arvioida sen perusteella, saiko käyttäjä asiansa hoidettua tehokkaasti ja vähäisellä kuormituksella.

Tekoäly nousi aineistossa uutena tekijänä, joka saattoi sekä lisätä tehokkuutta että synnyttää uusia kestävyyshaasteita. Kun se nopeutti kehitystyötä, se saattoi samalla kasvattaa laskentaresurssien tarvetta, kehittäjäkohtaista ympäristökuormaa ja huonolaatuisen koodin riskiä.

> ”Tällä hetkellä vastaan tulee myös vastakkaisia trendejä, kuten tekoäly, joka kasvattaa kehittäjäkohtaista ympäristökuormaa. En tiedä, tuoko se vastaavan määrän tehokkuutta mukaan niin, että kokonaisvaikutus jäisi lähelle nollaa vai ei. Arviointi muuttuu koko ajan vaikeammaksi, koska enää ei voida ajatella, että kehittäjä kuluttaa hyvin vähän resursseja suhteessa palvelun ajokustannuksiin.” (H5)

> ”AI on tällä hetkellä niin iso buumi, että kaikki pitäisi tehdä nopeasti ja halvalla. Ihmiset myyvät sovelluskehitystä paljon pienemmällä marginaalilla kuin ennen, koska asioita tehdään AI:lla. Siinä kuitenkin unohdetaan se, että AI tekee hyvin usein ihan mitä sattuu. Tällä hetkellä on siis vaikeaa tehdä kestävää kehitystä ja olla samalla kilpailukykyinen. […] Haasteena on kuitenkin se, että tekoälyn tekemään koodiin luotetaan välillä liikaa. Jos taitava kehittäjä ei käy tekoälyn tuottamaa koodia läpi — tai jos siihen ei ole edes aikaa — se voi heikentää kestävyyttä merkittävästi.” (H1)

Tekoälyyn liittyi sekä rebound-tyyppinen että laadullinen riski, kuten yllä kuvattiin. Tekoäly saattoi lisätä laskentaa ja energiankulutusta, mutta myös tuottaa vaikeasti ylläpidettävää koodia, jos sen tuloksia ei arvioitu riittävästi.

Tekoälyn käyttöön liittyi myös tarve määritellä ohjelmistokehitykselle laadullinen vähimmäistaso.

> ”Olisi välillä ihan hyväkin, että tulisi tietyt vaatimukset: uudella ohjelmistolla pitäisi olla jokin minimitaso. Jotkin kriteerit pitäisi täyttyä, ettei kaikki paikat ole vain täynnä AI-generoitua puppua. Onko se hyödyllistä ja kestävää? Tuskin.” (H5)

Kestävyyden arviointi ei perustunut vain tuotannon nopeuteen tai välittömään kustannukseen. Myös ylläpidettävyys, laatu ja elinkaaren aikaiset vaikutukset oli huomioitava. Haitallisten käytönaikaisten vaikutusten tunnistamista vaikeutti myös se, ettei kehittäjällä tai toimittajalla aina ollut näkyvyyttä pilvikustannuksiin, käyttöasteeseen tai resurssien kulutukseen.

> ”Siinä tulee myös sellainen problematiikka, että kun kyseessä on konsulttitalo, konsultteilla ei välttämättä ole näkyvyyttä pilvikustannuksiin ja pilven käyttöön.

> Se tieto voi tulla asiakkaan organisaation muulta taholta. Tämä on sellainen asia, johon olen usein törmännyt. Haluaisi nähdä ja optimoida, mutta siihen ei ole pääsyä. Konsulttien rooli voi olla se, että tuotamme koodia ja laitamme sen ajoon. Palveluntarjoajilla on kyllä hyviä raportteja, jotka tekevät myös analyysiä. Ne voivat esimerkiksi kertoa, että jokin palvelu on vajaalla käytöllä.” (H5)

Vaikutuksia ei voitu aina tunnistaa, vaikka asiantuntijalla olisi ollut halua optimoida järjestelmää. Ilman näkyvyyttä käyttöön ja resurssien kulutukseen kädenjäljen ja haitallisten vaikutusten arviointi jäi epäsuoraksi.

Ohjelmistojen kädenjälki, haitalliset käytönaikaiset vaikutukset ja rebound-tyyppiset riskit liittyivät siihen, vähensikö ohjelmisto aidosti kokonaiskuormitusta vai siirsikö tai kasvattiko se sitä. Positiivinen kädenjälki syntyi, kun ohjelmisto vähensi hukkaa, sujuvoitti toimintaa ja tuki pitkäikäistä ratkaisua. Haitallinen vaikutus syntyi, jos ohjelmisto lisäsi käyttäjän työtä, ylläpiti tarpeetonta koodia tai järjestelmiä, kasvatti kokonaiskäyttöä ilman todellista hyötyä tai tuotti vaikeasti ylläpidettävää ratkaisua. Rebound-vaikutuksia ei käsitelty haastatteluissa vakiintuneena käsitteenä, mutta aineistosta voitiin tunnistaa rebound-tyyppisiä tilanteita erityisesti palvelun käytön kasvun ja tekoälyn käytön yhteydessä.

4.6 Mittarit, työkalut ja arviointikäytännöt

Haastatteluaineiston perusteella ohjelmistokehityksen kestävyyden mittaaminen ja arviointi olivat vielä keskeneräisiä. Organisaatioissa oli käytössä teknisiä mittareita, seurantatyökaluja ja arviointikäytäntöjä, mutta ne kohdistuivat useimmiten suorituskykyyn, käytettävyyteen, pilviresurssien käyttöön, kustannuksiin, tietoturvaan tai ylläpidettävyyteen. Varsinainen jatkuva kestävyyden, energiankulutuksen tai ohjelmistokohtaisen hiilijalanjäljen mittaaminen eivät olleet vielä vakiintunut käytäntö.

Mittarit näkyivät aineistossa ensisijaisesti teknisinä ja operatiivisina mittareina. Esimerkiksi CPU-tasot, kuormitusrajat, vasteajat, SLA-rajat ja hälytykset auttoivat seuraamaan järjestelmän toimintaa ja resurssien käyttöä, mutta niitä ei yleensä tulkita suoraan kestävyyden mittareiksi.

> ” Niissä tuotteissa, joita olen itse nähnyt, on mielestäni jonkinlaisia mittareita, ja niitä kyllä tutkitaan tiettyyn pisteeseen asti. Omalla puolellani suuria mittareita ei kuitenkaan ole. Käytössä ovat lähinnä pilvipalveluiden mittarit, joista näkyvät esimerkiksi CPU-tasot. Niiden kautta voidaan tarkastella energiankulutusta, mikä taas kertoo, mitä meidän pitäisi muuttaa, jotta kulutusta saataisiin pienemmäksi. Tämä on myös kustannuskysymys asiakkaalle: miten saamme pilvipalveluiden käytön ja kustannukset mahdollisimman tehokkaiksi.” (H1)

Kestävyyttä tukevia mittareita oli käytössä, mutta ne olivat pääosin epäsuoria. CPU-tasot, pilviresurssien käyttö ja kustannukset kertoivat ohjelmiston kuormituksesta, mutta ne eivät vielä muodostaneet varsinaista kestävyyden mittaristoa.

Suorituskykyyn ja käytettävyyteen liittyvät mittarit olivat vakiintuneempia kuin kestävyyden mittarit. Niitä seurataan palvelutasovaatimusten ja pilviympäristöjen hälytysten avulla, koska niille oli asetettu selkeitä rajoja.

> ”Yleensä suorituskykyyn liittyvät asiat tulevat kriteereistä. Esimerkiksi sovellukselle voidaan määritellä, millainen vastausaika sillä saa olla tai kuinka paljon se saa olla alhaalla vuodessa. Nämä ovat perus-SLA-rajoja. Käytännössä näitä mittareita seurataan, koska rajat eivät saa ylittyä. Sovelluksen täytyy siis vastata riittävän nopeasti ja pysyä sovitulla tavalla käytettävissä. Muuta varsinaista mittaamista ei ehkä samalla tavalla ole. Suorituskykymittareita seurataan kuitenkin suoraan pilvipalveluiden mittareista.” (H1)

Kestävyydeltä vastaavat rajat, tavoitteet ja hyväksymiskriteerit puuttuivat. Tämän vuoksi kestävyys jäi helposti teknisen suorituskyvyn ja kustannustehokkuuden sivuvaikutukseksi.

Pilvikustannusten seuranta puolestaan toimi käytännössä yhtenä epäsuorana kestävyyden arvioinnin välineenä, koska kustannukset liittyivät usein käytettyihin resursseihin.

> ”Yksi selkeä mittari on esimerkiksi palvelinten käytössä se, että mennään AWS:n Cost Exploreriin tai vastaavaan työkaluun ja katsotaan, paljonko tämä maksaa. Se on yleensä melko suoraan verrannollinen siihen, kuinka paljon palvelu kuluttaa sähköä. Tämä on tietenkin hyvin karkea kuva, mutta ainakin tällä tarkkuudella asiakas seuraa sitä yleensä hyvin tarkasti.” (H4)

Kustannusmittarit olivat aineiston perusteella vahvemmassa asemassa kuin varsinaiset ympäristömittarit. Ne pystyivät ohjaamaan resurssitehokkuuteen, mutta kustannusta ei nähty samana asiana kuin ympäristövaikutusta.

Joissakin pilviympäristöissä oli saatavilla CO_2-ekvivalenttiarvioita, mutta niitä ei aineiston perusteella seurattu jatkuvasti projektien ohjauksessa.

> ”Monista pilviympäristöistä saa myös jonkinlaisen arvion siitä, kuinka paljon CO_2-ekvivalenttia jokin palvelu on käyttänyt tietyssä kuukaudessa. En kuitenkaan tiedä projektia, jossa tätä seurattaisiin kuukausitasolla jatkuvasti niin, että sen seuraaminen olisi varsinainen tarkoitus.” (H4)

CO_2-arvio antoi suuntaa ympäristövaikutuksista, mutta ilman säännöllistä seurantaa, vertailua ja päätöksentekoon kytkemistä se jäi irralliseksi tiedoksi.

Mittareiden rinnalla tarvittiin laadullisia arviointikäytäntöjä. Checklist-tyyppiset työkalut saattoivat auttaa projektitiimiä tunnistamaan kestävyyteen liittyviä näkökulmia ja muuttamaan niitä konkreettisiksi tehtäviksi.

> ”Tunnistamiseen sanoisin, että tähän asti se on ollut pitkälti aika ad hoc -tyyppistä. Ihmiset tunnistavat näitä asioita oman asiantuntemuksensa perusteella. Tämä on tietenkin vähän hankalaa siinä mielessä, että jos projektiin ei esimerkiksi ole tilattu palvelumuotoilua, siellä saattaa jäädä joitakin asioita huomioimatta. Silloin saatetaan huomioida vain esimerkiksi teknisiä yksityiskohtia. Meidän kestävyystiimimme kanssa tekemämme checklist on kuitenkin sellainen, jonka projektitiimi pystyy tekemään itsenäisesti. Siinä käydään läpi erilaisia näkökulmia, joiden kautta näitä asioita voidaan tunnistaa.” (H4)

Kestävyyden tunnistaminen oli ilman yhteistä työkalua henkilöriippuvaista. Checklist saattoi laajentaa tarkastelua ja tuoda kestävyyden eri näkökulmat projektitiimin käyttöön.

Osa kestävyyden arvioinnista perustui myös dokumentaatioon, suunnitelmiin ja auditointiin. Tällöin kyse ei ollut yksittäisestä numeerisesta mittarista, vaan siitä, voitiinko esimerkiksi ylläpidettävyys, päivitysmalli ja pitkäikäisyys osoittaa käytännöillä ja suunnitelmilla.

> ”Ne ovat ehkä vähän tulkinnanvaraisia. En ole ainakaan törmännyt selkeään asteikkoon, jossa sanottaisiin, että tiettyyn tasoon on päästävä. Sen sijaan pitää pystyä osoittamaan esimerkiksi dokumentaatiolla tai muulla tavalla, että nämä asiat on huomioitu. Sitä on ehkä vaikea arvioida puhtaasti jollakin yksittäisellä mittarilla. Arviointi pohjautuu enemmän käytäntöihin ja siihen, millaisia suunnitelmia on tehty.” (H5)

Tämä korosti, että ohjelmiston kestävyys oli sekä mitattava että laadullisesti arvioitava ilmiö. Mittaristo ei ollut millään tavalla yksiselitteinen eikä käytänteet vakiintuneita.

Aineistossa nousi esiin myös tarve tehdä kestävyys näkyväksi kaikissa projekteissa ja vertailla mittareita projektien välillä.

> ”Keskeistä olisi mittarointi. Kestävyyden pitäisi olla ainakin näkyvissä kaikissa projekteissa jollakin tasolla. Esimerkiksi pitäisi saada tietää, että meillä on tällainen projekti, jossa tehdään palvelua, joka palvelee tiettyä määrää asiakkaita kuukaudessa, ja sen CO_2-ekvivalentti on tietyn suuruinen. Silloin pystyttäisiin vertailemaan CO_2-ekvivalenttia per kuukausi per projekti. Näin saataisiin jonkinlainen kuva siitä, onko kyse paljon vai vähän. Yhden projektin sisällä yhdestä numerosta ei vielä voi sanoa, onko se oikeasti paljon vai vähän. Sitä pitäisi pystyä vertailemaan projektien välillä.” (H5)

Yksittäinen mittari ei riittänyt, ellei sitä suhteutettu esimerkiksi käyttömäärään, asiakasmäärään, aikaan tai muihin projekteihin.

Mittaamisen lisäksi tarvittiin säännöllinen arviointiprosessi, jossa nykytilaa, muutoksia ja jatkotoimenpiteitä tarkastellaan projektin aikana. Muutoin mittaaminen jää kertaluontoiseksi, eikä vertailukohtia tai kokonaiskuvaa synny.

> ”Lisäksi tarvittaisiin sitä, että esimerkiksi meidän checklistiä tai muita vastaavia tapoja käytettäisiin näiden teemojen huomioimiseen myös silloin, kun tekijät eivät itse ole asiantuntijoita. Näitä asioita pitäisi tarkastella tietyin aikavälein: miten ne on huomioitu ja mikä niiden tilanne on. Tämä vaatisi myös sen, että asiakas sanoisi haluavansa näiden asioiden olevan huomioituja. Silloin voitaisiin päästä esimerkiksi siihen, että järjestetään puolivuosittainen työpajapäivä, jossa käydään checklistit läpi, katsotaan missä tilassa ollaan ja miten tilanne on muuttunut edellisestä kerrasta. Se ei välttämättä olisi edes mahdottoman suuri panostus, mutta sen avulla tiedettäisiin, missä mennään. Se olisi oikea alku: tiedostetaan nykytila ja suunta.” (H4)

Kestävyyden mittaaminen ja arviointi ovat aineiston perusteella ohjelmistokehityksessä vielä keskeneräisiä. Käytössä olevat tekniset ja taloudelliset mittarit tukivat kestävyyden arviointia epäsuorasti, mutta ne eivät vielä muodostaneet systemaattista kestävyyden mittaristoa. CO_2-ekvivalenttiarvioita on saatavilla, mutta niitä ei seurattu vakiintuneesti projektien ohjauksessa. Numeeristen mittareiden rinnalle kaivattiin laadullisia arviointikäytäntöjä, kuten checklistejä, dokumentaatiota, auditointeja ja säännöllisiä arviointityöpajoja. Kestävyyden systemaattinen integrointi edellytti mittareiden lisäksi prosessia, jossa havainnot kytketään vastuisiin, seurantaan, backlog-tehtäviin ja päätöksentekoon.

4.7 Organisaation vastuut, osaaminen ja kypsyys

Kestävyyden integrointi ohjelmistokehitykseen liittyy organisaation vastuisiin, osaamiseen, päätöksentekoon ja projektien ohjaukseen. Haastatteluaineiston perusteella kestävyys tunnistettiin organisaatioissa, mutta vastuut ja toimintamallit olivat usein epäselviä. Kestävyys saattoi näkyä yrityksen yleisessä vastuullisuustyössä, eettisissä periaatteissa tai yksittäisten asiantuntijoiden aktiivisuutena, mutta ei siirtynyt ohjelmistokehityksen projektitasolle.

> ”Meillä on Chief Sustainability Officer, […] siinäkin puhutaan lähinnä yleisestä vastuullisuudesta. Meillä ei ole minkäänlaista keskitettyä kontrollia esimerkiksi siitä, kuinka kestävää meidän ohjelmistokehityksemme on. […] Muuten se jää yksittäisen kehittäjän vastuulle, jos hän ottaa sen esiin.” (H3)

Vastuullisuudelle oli organisaatiotasolla omistaja, mutta ohjelmistokehityksen kestävyydelle ei silti ollut selkeää vastuuroolia. Vastuu jäi tällöin yksittäisille kehittäjille tai asiantuntijoille, joilla ei ollut muodollista valtaa vaikuttaa projektin tavoitteisiin, vaatimuksiin tai resursointiin.

Aineistossa erottui myös organisaation oman vastuullisuustyön ja asiakasprojektien välinen ero. Organisaatio saattoi olla kehittynyt esimerkiksi hiilijalanjäljen raportoinnissa, mutta asiakasprojekteissa vastuu jäi helposti määrittelemättä.

> ”Meillä on esimerkiksi aika hyvää omaa kestävyysjalanjälkiraportointia olemassa, ja olemme saaneet asiasta sertifikaatin. […] Mutta asiakkaiden projekteissa meillä ei ole samalla tavalla vastuuhenkilöitä juuri siksi, että vastuu pitäisi määritellä projektikohtaisesti tai asiakaskohtaisesti.” (H4)

Kestävän ohjelmistokehityksen vastuu toivottiin määriteltävän erikseen projektitasolla. Kun vastuuta ei jaettu asiakkaan, toimittajan, projektipäällikön, arkkitehdin tai kehitystiimin välillä, kestävyys jäi helposti yleiseksi periaatteeksi.

Kestävyysosaaminen oli aineiston perusteella poikkitieteistä ja roolien välistä. Se saattoi liittyä arkkitehtuuriin, pilvipalveluihin, etiikkaan, sustainability-tiimeihin, palvelumuotoiluun, analytiikkaan ja organisaation sisäiseen osaamisen kehittämiseen. Tämä tarkoitti, että kestävä ohjelmistokehitys edellytti teknisen osaamisen lisäksi myös käyttäjien, prosessien, vastuullisuuden ja päätöksenteon ymmärtämistä.

Organisaatioiden kypsyys suhteessa kestävyyteen vaihteli aineistossa selvästi. Joissakin tapauksissa kestävyys ei vielä näkynyt ohjelmistokehityksen käytännöissä juuri lainkaan, vaikka se olisi yleisellä tasolla tunnistettu.

> ”Tämän läpikäydyn perusteella organisaatiota ei voi pitää kovin kypsänä. Jos asiaa ei ole oikein aloitettukaan, niin ei se silloin ole kypsää.” (H2)

Kypsyys ei tarkoittanut pelkkää tietoisuutta kestävyyden merkityksestä. Kypsyys edellytti, että kestävyyttä oli alettu käsittelemään käytännön toimintana ja että se oli kytketty ohjelmistokehityksen prosesseihin.

Toisaalta aineistossa oli myös organisaatioita, joissa osaamista ja tietoisuutta oli jo enemmän. Tämä näkyy sisäisinä asiantuntijoina, koulutuksina, sertifikaatteina, oppaina, sustainability-tiimeinä ja eettisinä arviointikäytäntöinä.

> ”Henkilökohtaisesti sanoisin, että oma osaamiseni on melko korkealla tasolla. Minulla on ehkä vielä isompia aukkoja tekoälyn vaikutuksiin liittyen, koska siitä alkaa jo olla jonkin verran dataa, sekä datakeskuksiin ja dataan liittyvissä kysymyksissä. Sanoisin kuitenkin, että olen huomattavasti tietoisempi näistä asioista kuin keskivertokehittäjä, ainakin meillä. Jos vertaan organisaatiota kilpailijoihin, sanoisin, että olemme aika samalla tasolla kuin suurin osa.” (H3)

Osaaminen ei jakautunut tasaisesti organisaation sisällä. Yksittäinen asiantuntija saattoi olla selvästi keskivertokehittäjää tietoisempi, vaikka organisaation kokonaiskypsyys olisi vielä ollut keskeneräinen.

Kokonaisuutena aineistosta voitiin päätellä, että kestävän ohjelmistokehityksen kypsyys on vielä kehittymässä. Kestävyys tunnistettiin, mutta se ei ollut useimmissa tapauksissa institutionalisoitunut ohjelmistokehityksen normaaliksi toimintamalliksi. Kypsyys edellytti, että kestävyys vietiin organisaatiotasolta projektitasolle: vaatimuksiin, mittareihin, arkkitehtuuripäätöksiin, vastuisiin, backlogille vietäviin tehtäviin, ylläpitoon ja päätöksentekoon. Kestävyys ei saanut jäädä yksittäisten kehittäjien tai arkkitehtien kiinnostuksen varaan, vaan se tarvitsi selkeän omistajuuden ja paikan projektien normaalissa ohjausmallissa.

4.8 Esteet, ristiriidat ja kehitystarpeet

Haastatteluaineiston perusteella kestävyyden huomioimista ohjelmistokehityksessä estivät erityisesti taloudelliset, organisatoriset, tekniset ja vaatimustenhallintaan liittyvät tekijät. Esteet eivät ensisijaisesti liittyneet kestävyyden merkityksen puutteelliseen tunnistamiseen, vaan siihen, että kestävyyttä ei ollut vielä operationalisoitu selkeiksi vaatimuksiksi, vastuiksi, mittareiksi ja resursoiduiksi käytännöiksi ohjelmistokehityksen prosesseissa.

Keskeinen aineistossa toistuva este oli asiakkaan vaatimusten ja maksuhalukkuuden puute. Asiakas- ja konsulttiprojekteissa kestävyys toteutui usein vain silloin, kun se oli mukana tarjouspyynnössä, hyväksymiskriteereissä tai projektin rahoituksessa. Jos kestävyyttä ei määritelty projektin alkuvaiheessa, se näyttäytyi myöhemmin helposti ylimääräisenä työnä eikä osana laatua.

> ”Kun asiakas ei ole valmis maksamaan siitä, siihen ei haluta investoida millään tavalla” (H3).

Toinen keskeinen este oli kiire ja aikataulupaine. Kestävyyden kaltaiset pitkän aikavälin näkökulmat jäivät helposti toissijaisiksi, vaikka juuri projektin alkuvaiheessa tehtiin ratkaisuja, jotka vaikuttivat ohjelmiston elinkaareen, ylläpidettävyyteen ja resurssien käyttöön.

> ”Yleensä on aina kiire. En ole ollut kertaakaan projektissa, jossa ei olisi ollut kiire.” (H3)

> ”Suuressa osassa näistä yhteinen tekijä on pitkälti asiakkaan raha ja sen pohjalta tehtävät päätökset tavalla tai toisella.” (H5)

Kestävyyttä rajoittivat usein samat tekijät kuin muutakin ohjelmistokehitystä: raha, aika ja projektin rajaukset. Kestävyys jäi sivuun, jos sitä ei nähty asiakkaalle välittömänä hyötynä tai jos sen arviointiin ei ollut varattu aikaa.

Kolmas este liittyi vastuiden epäselvyyteen. Kestävyys jäi asiakkaan, toimittajan ja yksittäisten asiantuntijoiden väliseksi jaetuksi, mutta nimeämättömäksi vastuuksi. Tällöin kukaan ei välttämättä omistanut kestävyyttä projektissa, vaikka sen merkitys tunnistettiin yleisellä tasolla.

> ”Sellaista selkeää henkilöä ei ole, eikä myöskään varsinaisia vaatimuksia.” (H5)

Vastuun puuttuminen liittyi suoraan vaatimusten puuttumiseen. Jos kestävyydelle ei ollut nimettyä vastuuta, se ei myöskään helposti muuttunut projektin ohjauksen, backlog-tehtävien, hyväksymiskriteerien tai arkkitehtuuripäätösten osaksi.

Neljäs este oli mittareiden ja arviointikäytäntöjen keskeneräisyys. Aineiston perusteella ongelma oli kuitenkin selvä: ilman mittareita ja arviointikäytäntöjä kestävyyttä oli vaikea verrata muihin projektitavoitteisiin, perustella asiakkaalle, priorisoida backlogilla tai seurata elinkaaren aikana.

Viides este liittyi osaamisen epätasaiseen jakautumiseen ja teknologisten vaikutusten näkymättömyyteen. Tekninen kestävyys, suorituskyky ja ylläpidettävyys olivat kehittäjille tuttuja, mutta niiden yhteys ympäristövaikutuksiin, kädenjälkeen, rebound-vaikutuksiin ja systeemisiin vaikutuksiin ei ollut selvä. Myös teknologinen abstraktio vaikeutti vaikutusten tunnistamista, koska kehittäjä ei välttämättä nähnyt suoraan, miten koodi, pilviresurssit, datansiirto tai ajonaikainen kuormitus vaikuttivat kokonaisenergiankulutukseen.

Kuudes este liittyi sääntelyn, standardien ja hankintakriteerien puutteeseen. Saavutettavuuteen ja tietoturvaan verrattuna kestävyydellä ei vielä ollut yhtä vakiintunutta asemaa ohjelmistokehityksen vaatimuksena. Tämä tarkoittaa, että kestävyyden huomioiminen riippuu usein asiakkaan omasta tahtotilasta, organisaation kypsyydestä tai yksittäisten asiantuntijoiden aktiivisuudesta.

Seitsemäs este liittyi elinkaaren loppupään hallintaan. Aineiston perusteella vanhojen järjestelmien, tarpeettomien toiminnallisuuksien ja käyttämättömien resurssien hallinta oli

olennainen kehityskohde. Jos järjestelmiä ei ajettu hallitusti alas, ne lisäsivät teknistä velkaa, ylläpitokuormaa ja resurssien kulutusta.

Keskeiset kehitystarpeet kohdistuivat kestävyyden systematisointiin. Ensimmäinen kehitystarve oli kestävyyden tuominen projektin alkuvaiheeseen, jossa määritellään tavoitteet, rajaukset, hyväksymiskriteerit ja vaikutukset. Toinen kehitystarve oli kestävyyden muuttaminen konkreettisiksi vaatimuksiksi, arkkitehtuuriperiaatteiksi, mittareiksi ja backlog-tehtäviksi. Kolmas kehitystarve oli vastuunjaon selkeyttäminen asiakkaan, toimittajan, projektipäällikön, arkkitehdin, kehittäjän, tuoteomistajan ja loppukäyttäjän välillä.

Lisäksi kaivattiin osaamisen kehittämistä, hankinta- ja sopimuskäytäntöjen tarkentamista sekä elinkaaren loppupään parempaa hallintaa. Kestävyyttä haluttiin tarkastella sekä määrällisten mittareiden että laadullisten arviointikäytäntöjen avulla, koska kaikki vaikutukset eivät näkyneet yksittäisinä teknisinä lukuina. Erityisesti kädenjäljen, rebound-vaikutusten ja systeemisten vaikutusten arviointi edellytti myös prosessien, käyttäjäpolkujen ja arvovirtojen tarkastelua.

Kokonaisuutena aineisto osoitti, että kestävyyden esteet muodostivat toisiinsa kytkeytyvän kokonaisuuden. Asiakkaan vaatimusten puute, kustannus- ja aikataulupaineet, mittareiden puute, vastuiden epäselvyys, osaamisvaje, teknologinen abstraktio, sääntelyn puute ja elinkaaren loppupään laiminlyönti vahvistivat toisiaan. Kestävyys tunnistettiin jo monissa organisaatioissa, mutta se ei vielä ollut vakiintunut osa ohjelmistokehityksen rakenteita. Aineistossa kävi ilmi, että tarvittaisiin yhtenäinen toimintamalli, joka auttaisi tunnistamaan vaikutukset, määrittelemään kestävyyteen liittyvät ei-toiminnalliset vaatimukset, mittaamaan ja arvioimaan vaikutuksia sekä jakamaan vastuut eri sidosryhmien välillä.

4.9 Yhteenveto empiirisistä havainnoista

Tässä luvussa tarkasteltiin H1–H5-haastatteluaineiston keskeisiä havaintoja suhteessa tutkimuksen teoreettiseen viitekehykseen ja tutkimuskysymyksiin. Aineiston perusteella kestävyys tunnistettiin ohjelmistokehityksessä merkitykselliseksi näkökulmaksi, mutta sen käytännön asema oli vielä epäyhtenäinen, projektikohtainen ja usein epäsuora. Kestävyys näkyi haastatteluissa erityisesti teknisen pitkäikäisyyden, ylläpidettävyyden, resurssitehokkuuden, kustannusten, sosiaalisen vastuun ja eettisten vaikutusten kautta. Ympäristöllinen kestävyys tunnistettiin, mutta se ei vielä ollut yhtä vakiintunut osa ohjelmistokehityksen arkea kuin tekninen ja taloudellinen kestävyys.

Aineisto osoittaa, että kestävyyttä voidaan tarkastella ohjelmiston laatuattribuuttina, mutta sen muuttaminen selkeäksi ei-toiminnalliseksi vaatimukseksi on organisaatioissa vielä keskeneräistä. Käytännössä kestävyys ei kuitenkaan vielä ollut vakiintunut eksplisiittiseksi NFR-vaatimukseksi, vaan se jäi usein teknisen laadun, kustannustehokkuuden, arkkitehtuuriperiaatteiden, dokumentaation, auditoinnin tai yksittäisten asiantuntijoiden harkinnan varaan. Kestävyys muuttui konkreettiseksi vaatimukseksi lähinnä silloin, kun asiakas, kilpailutus, auditointi, sääntely tai organisaation oma toimintamalli sitä edellytti.

Elinkaaren näkökulmasta kestävyys tunnistettiin vahvimmin ohjelmistokehityksen alkuvaiheissa, erityisesti määrittelyssä, suunnittelussa ja arkkitehtuurissa. Näissä vaiheissa tehdään ratkaisuja, jotka vaikuttavat ohjelmiston pitkäikäisyyteen, ylläpidettävyyteen, teknologioiden elinkaareen, infrastruktuurivalintoihin, pilviympäristöihin ja resurssien käyttöön. Toteutus- ja testausvaiheessa kestävyys näkyi useammin välillisesti suorituskyvyn, resurssitehokkuuden, skaalautuvuuden ja kustannusten hallinnan kautta. Ylläpidossa ja käytöstä poistossa kestävyys liittyi teknisen velan hallintaan, päivityskäytäntöihin, tarpeettomien ominaisuuksien poistamiseen ja vanhojen järjestelmien hallittuun alasajoon.

Haastatteluaineisto tuki myös ohjelmistojen suorien, epäsuorien ja systeemisten vaikutusten erottelua. Suorat vaikutukset liittyivät haastatteluissa pilviresursseihin, datakeskuksiin ja koodin tehokkuuteen, energiankulutukseen ja infrastruktuuriin. Epäsuorat vaikutukset näkyivät manuaalisen työn vähenemisenä, prosessien muutoksina, itsepalveluna tai käyttäjän työn lisääntymisenä. Systeemiset vaikutukset liittyivät laajempiin palvelurakenteiden ja käyttäytymisen muutoksiin sekä toimintatapojen muutoksiin. Aineiston perusteella haastateltavat tunnistivat näitä vaikutuksia käytännön esimerkkien kautta, vaikka he eivät haastatteluissa käyttäneetkään käsitteitä suora, epäsuora ja systeeminen vaikutus, kuten teoriassa ne käsiteltiin.

Haastatteluaineistossa ohjelmistojen kädenjälkivaikutukset tunnistettiin myönteisinä vaikutuksina työn sujuvoittamisen, resurssien säästämisen, manuaalityön vähentämisen, käyttäjäpolkujen optimoinnin ja palveluprosessien parantamisen kautta. Samalla aineisto osoitti, ettei digitalisaatio automaattisesti tuota positiivista kädenjälkeä. Ohjelmisto voi myös lisätä käyttäjän työtä, kasvattaa kokonaiskäyttöä, ylläpitää tarpeetonta teknistä velkaa tai siirtää kuormitusta käyttäjille, ylläpitoon tai infrastruktuuriin. Rebound-vaikutukset eivät esiintyneet haastatteluissa vakiintuneena käsitteenä, mutta aineistosta voitiin tunnistaa rebound-tyyppisiä riskejä erityisesti palvelun käytön kasvun ja tekoälyn käytön yhteydessä.

Mittareiden ja arviointikäytäntöjen osalta aineisto osoitti, että kestävyyden mittaaminen on vielä keskeneräistä. Organisaatioissa oli käytössä teknisiä ja taloudellisia mittareita, kuten pilviresurssien käyttö, CPU-tasot, vasteajat, SLA-rajat, hälytykset ja kustannusseuranta. Näitä voitiin käyttää kestävyyden epäsuorina indikaattoreina, mutta ne eivät vielä muodostaneet varsinaista kestävyyden mittaristoa. CO_2-ekvivalenttiarvioita on joissakin pilviympäristöissä saatavilla, mutta niiden jatkuva käyttö projektien ohjauksessa ei ole vakiintunut. Aineiston perusteella määrällisten mittareiden rinnalle tarvitaan laadullisia arviointikäytäntöjä, kuten checklistejä, dokumentaatiota, auditointeja ja säännöllisiä arviointityöpajoja.

Organisaation vastuiden ja kypsyyden osalta aineistosta käy ilmi, että kestävyys tunnistettiin monissa organisaatioissa, mutta vastuut ja toimintamallit olivat edelleen epäselviä. Organisaatiolla oli yleistä vastuullisuustyötä, vastuullisuusraportointia tai sustainability-rooleja, mutta ohjelmistokehityksen projektitasolla kestävyydelle ei välttämättä ollut selkeää omistajaa. Vastuu jäi helposti yksittäisten kehittäjien, arkkitehtien tai asiantuntijoiden kiinnostuksen varaan. Kestävyyden huomioiminen nähtiin näin henkilöriippuvaiseksi, mikä vaikeutti sen systemaattista jalkauttamista.

Keskeiset esteet liittyivät asiakkaan vaatimusten ja maksuhalukkuuden puutteeseen, kustannus- ja aikataulupaineisiin, mittareiden puutteeseen, vastuiden epäselvyyteen, osaamisen epätasaiseen jakautumiseen, teknologiseen abstraktioon, sääntelyn puutteeseen ja elinkaaren loppupään laiminlyöntiin. Esteet eivät ensisijaisesti liittyneet kestävyyden merkityksen tunnistamattomuuteen, vaan siihen, ettei kestävyyttä oltu operationalisoitu selkeiksi vaatimuksiksi, vastuiksi, mittareiksi ja resursoiduiksi käytännöiksi ohjelmistokehityksen prosesseissa.

Kokonaisuutena empiirinen aineisto vahvisti teoriakatsauksessa esitellyn aikaisemman kirjallisuuden, jonka mukaan kestävyys on siirtymässä ohjelmistokehityksessä yleisestä vastuullisuuspuheesta kohti käytännön kehityskysymystä, mutta tämä siirtymä oli edelleen kesken. Kestävyys tunnistettiin, mutta se ei ollut vakiintunut osaksi vaatimustenhallintaa, arkkitehtuuripäätöksiä, mittareita, backlog-tehtäviä, projektiohjausta ja elinkaaren hallintaa. Tämän vuoksi tutkimuksen seuraavassa luvussa esitettävän kestävyysmallin lähtökohtana on tarve jäsentää kestävyyden huomioiminen ohjelmistokehityksessä systemaattisesti: mitä vaikutuksia arvioidaan, missä elinkaaren vaiheessa arviointi tehdään, millä mittareilla ja käytännöillä arviointia tuetaan sekä kenellä on vastuu kestävyyden huomioimisesta.

Pääkoodi	Alakoodit	Empiirinen havainto H1–H5	Alaluku	Tutkimus-kysymys
K1 Kestävyyden käsitteellistäminen	K1.1–K1.7	Kestävyys ymmärretään moniulotteisesti, mutta tekninen kestävyys painottuu vahvimmin. Ympäristöllinen ja sosiaalinen kestävyys tunnistetaan, mutta niiden operationalisointi on heikompaa.	4.1	TK1
K2 Kestävyys ei-toiminnallisena vaatimuksena	K2.1–K2.7	Kestävyys tunnistetaan mahdollisena laatuattribuuttina, mutta se on harvoin eksplisiittinen NFR-vaatimus. Usein se jää implisiittiseksi teknisen laadun osaksi.	4.2	TK1
K3 Kestävyys elinkaaren vaiheissa	K3.1–K3.9	Kestävyys tunnistetaan parhaiten määrittelyssä, suunnittelussa ja arkkitehtuurissa, mutta testaus, käyttöönotto, ylläpito ja käytöstä poisto ovat hajanaisempia.	4.3	TK1
K4 Suorat, epäsuorat ja systeemiset vaikutukset	K4.1–K4.9	Suorat vaikutukset tunnistetaan resurssien ja infrastruktuurin kautta. Epäsuorat ja systeemiset vaikutukset tunnistetaan prosessien ja käyttäjävaikutusten kautta, mutta ei systemaattisesti.	4.4	TK2
K5 Kädenjälki, rebound ja haitalliset vaikutukset	K5.1–K5.9	Ohjelmistot voivat tehostaa toimintaa ja vähentää hukkaa, mutta ne voivat myös lisätä kuormitusta, teknistä velkaa ja rebound-vaikutuksia.	4.5	TK2
K6 Mittarit, työkalut ja arviointikäytännöt	K6.1–K6.10	Mittaaminen on hajanaista. Käytössä on kustannus-, suorituskyky-, pilvimittareita, mutta ei yhtenäistä kestävyyden mittaristoa.	4.6	TK3
K7 Vastuut, osaaminen ja kypsyys	K7.1–K7.9	Organisaatioissa on osaamista ja vastuullisuusrakenteita, mutta projektitason vastuut ovat epäselviä ja usein asiakas- tai henkilöriippuvaisia.	4.7	TK3, TK4
K8 Esteet, ristiriidat ja kehitystarpeet	K8.1–K8.11	Esteitä ovat kiire, kustannuspaine, asiakkaan vaatimusten puute, mittareiden puute, vastuiden epäselvyys, osaamisvaje, teknologinen abstraktio, sääntelyn ja standardien puute sekä elinkaaren loppupään laiminlyönti.	4.8	TK3, TK4
K9 Aineistosta nousevat uudet teemat	K9.1–K9.5	Aineistossa uusina tai vahvistuneina teemoina korostuvat erityisesti tekoäly, käytöstä poisto, ajopaikan merkitys, auditointi, checklist-käytännöt ja vastuun jakautuminen asiakkaan, toimittajan ja kehitystiimin välillä.	4.1–4.8 ja 5	TK1-4

Taulukko 5. Empiiristen havaintojen yhteenveto ja yhteys tutkimuskysymyksiin

Taulukko 5 kokoaa luvun 4 keskeiset empiiriset havainnot ja osoittaa, miten ne liittyvät tutkimuskysymyksiin sekä luvussa 5 muodostettavaan kestävyysmalliin. Havaintojen perusteella kestävyyden systemaattinen huomioiminen edellyttää käsitteellistä jäsentämistä, elinkaaren mukaista tarkastelua, mittareita, laadullisia arviointikäytäntöjä sekä selkeää vastuunjakoa asiakkaan, toimittajan ja kehitystiimin välillä.

5 Kestävyysmalli

Tässä luvussa esitetään tutkimuksen teoreettisen viitekehyksen ja empiiristen havaintojen perusteella muodostettu kestävyysmalli ohjelmistokehitykseen. Mallin tarkoituksena on tukea ohjelmistokehitystiimejä, projektipäälliköitä, arkkitehtejä, tuoteomistajia, asiakkaita ja organisaation vastuullisuustiimejä tunnistamaan, arvioimaan ja hallitsemaan ohjelmistojen kestävyyteen liittyviä vaikutuksia nykyistä systemaattisemmin. Malli ei korvaa olemassa olevia ohjelmistokehityksen menetelmiä, vaan täydentää niitä kestävyyden näkökulmalla.

Ohjelmistojen kestävyyden arviointia ei voida tehdä ainoastaan koodin tehokkuuden tai energiankulutuksen perusteella. Empiirisen aineiston ja teoreettisen viitekehyksen perusteella ohjelmiston vaikutukset syntyvät koko elinkaaren aikana, ja sen systeemiset vaikutukset ovat joko suoria tai epäsuoria. Näistä voisi mainita infrastruktuurin kuormituksen ja käyttäjien työn muutokset, prosessien tehostuminen, teknisen velan kasvaminen tai palvelurakenteiden muuttuminen.

Tämän vuoksi malli yhdistää viisi erilaista näkökulmaa: kestävyyden ulottuvuudet, ohjelmistojen vaikutustasot, ohjelmistokehityksen elinkaarivaiheet (katso 2.3), arviointi- ja mittauskäytännöt sekä niihin liittyvän vastuunjaon (katso 2.2.7). Näitä osa-alueita analysoimalla voidaan tunnistaa kestävyyteen liittyviä vaikutuksia ohjelmistokehityksen eri vaiheessa, sekä kuinka niitä voidaan arvioida ja kenen vastuulla niiden huomioiminen on.

Mallin tavoitteena on tehdä ohjelmistokehityksen kestävyydestä osa vaatimustenhallintaa, arkkitehtuuripäätöksiä, teknisiä ratkaisuja, mittaamista, dokumentointia, ylläpitoa ja päätöksentekoa. Yleisesti se kestävyys tunnistetaan vain vastuullisuustavoitteena.

Seuraavissa alaluvuissa kuvataan ensin mallin lähtökohdat. Tämän jälkeen esitellään mallin perusrakenne ja sen soveltaminen ohjelmistokehityksen elinkaaren eri vaiheissa sekä vastuunjako eri sidosryhmien välillä.

5.1 Lähtökohdat

Mallin lähtökohtana on luvussa 4 esitetyt empiiriset havainnot: kestävyys tunnistetaan ohjelmistokehityksessä tärkeäksi näkökulmaksi, mutta sen käytännön asema on toistaiseksi epäyhtenäinen. Kestävyys näkyy haastatteluaineistossa erityisesti teknisen pitkäikäisyyden, ylläpidettävyyden, resurssitehokkuuden, kustannusten, sosiaalisen vastuun ja eettisten

vaikutusten kautta. Useimmissa tapauksissa se ei kuitenkaan ole selkeästi määritelty vaatimus, mittari tai projektin ohjauksen osa-alue. Tästä muodostuu kestävyysmallin keskeinen tarve.

Haastatteluaineiston perusteella kestävyys vaikuttaa jäävän teknisen laadun epäsuoraksi seuraukseksi ja sen toteutuminen riippuu liikaa yksittäisistä henkilöistä tai asiakkaan vaatimuksista. Tällöin kestävyyttä ei tunnisteta, dokumentoida eikä sen vaikutuksia arvioida yksiselitteisesti.

Tämän vuoksi ensimmäinen lähtökohta mallissa on kestävyyden käsitteleminen ohjelmiston laatuattribuuttina ja ei-toiminnallisena vaatimuksena. Kestävyys näkyy siinä, miten järjestelmä toimii, miten sitä käytetään ja ylläpidetään, kuinka paljon se kuluttaa resursseja ja millaisia vaikutuksia sillä on käyttäjiin ja ympäröivään toimintaan koko elinkaarensa aikana. Tältä osin se voidaan rinnastaa muihin ei-toiminnallisiin vaatimuksiin. Tämä tarkoittaa sitä, että kestävyyttä pitää pystyä määrittelemään ja arvioimaan samalla tavalla kuin muitakin järjestelmän laatuvaatimuksia.

Toisena lähtökohtana on koko ohjelmiston elinkaaren huomioiminen. GREENSOFT-mallin mukainen cradle-to-grave-ajattelu korostaa, että ohjelmiston kestävyyttä ei voida arvioida vain toteutusvaiheessa tai ohjelmiston ajonaikaisen energiankulutuksen perusteella (Naumann ym., 2014). Kestävyyteen vaikuttavat projektin alkuvaiheessa tehtävät päätökset: mitä järjestelmää ollaan rakentamassa, miksi se rakennetaan, millaisia prosesseja se muuttaa ja millaisia vaikutuksia siltä odotetaan. Myöhemmissä projektin vaiheissa ovat ratkaisevia arkkitehtuuri, teknologiavalinnat, toteutustapa, testaus, käyttöönotto, ylläpito ja käytöstä poisto. Elinkaariajattelu on erityisen tärkeää, koska ohjelmiston vipuvaikutukset realisoituvat usein vasta käytössä. Tekninen velka näkyy esimerkiksi turhina vanhoina järjestelminä, jotka pyörivät pilvessä tarpeettomina.

Kolmas lähtökohta liittyy vaikutustasojen erottamiseen. Teoreettisessa viitekehyksessä ICT:n vaikutukset jaetaan suoriin, epäsuoriin ja systeemisiin vaikutuksiin (Hilty & Aebischer, 2015). Haastatteluaineistossa näitä käsitteitä ei aina käytetty sellaisenaan, mutta näitä vastaavia havaintoja nousi esiin useissa yhteyksissä. Suorat vaikutukset liittyivät energiankulutukseen, pilviresursseihin, datakeskuksiin ja infrastruktuuriin. Epäsuorat vaikutukset liittyivät prosessien tehostumiseen, automaatioon, itsepalveluun ja käyttäjän työn muutoksiin. Systeemiset vaikutukset puolestaan liittyivät laajempiin toimintatapojen, palvelurakenteiden ja käyttäytymisen muutoksiin. Mallin avulla pyritään tunnistamaan nämä kaikki vaikutustasot.

Neljäs lähtökohta on positiivisen kädenjäljen ja haitallisen jalanjäljen tarkastelu samanaikaisesti. Ohjelmistot tuottavat myönteisiä kädenjälkivaikutuksia parantamalla työn sujuvuutta, vähentämällä hukkaa, vähentämällä resurssien käyttöä tai mahdollistamalla kestävämpiä toimintatapoja. Samaan aikaan on kuitenkin tunnistettava, että ohjelmisto mahdollisesti synnyttää myös kielteisiä vaikutuksia. Digitalisointi ei automaattisesti tarkoita kestävyyttä. Jos ratkaisu lisää kokonaiskuormitusta, käyttäjän työtä, laskentaa, ylläpitotarvetta tai teknistä monimutkaisuutta, sen vaikutus on usein kestävyyden kannalta kielteinen. Tämän vuoksi mallissa ei arvioida ainoastaan sitä, onko ohjelmisto teknisesti tehokas. Mallilla pyritään arvioimaan, mitä ohjelmisto saa aikaan sen toimintaympäristössään.

Viidentenä lähtökohtana nostetaan mittaamisen ja arvioinnin yhdistämisen merkitys. Empiirisen aineiston perusteella ohjelmistokehityksessä on käytössä useita teknisiä ja taloudellisia mittareita. Ne tukevat kestävyyden arviointia epäsuorasti, mutta niistä ei vielä muodostu yhtenäistä kestävyyden mittaristoa. Lisäksi kestävyyden kaikki vaikutukset eivät ole mitattavissa yhdellä numeerisella suureella.

Malliin tarvitaan siis sekä määrällisiä että laadullisia arviointikäytäntöjä. Määrällisinä mittareina voidaan pitää resurssien käyttöä, kustannuksia, kuormitusta, energiankulutusta tai CO_2-ekvivalenttiarvioita. Laadullisina arviointikäytäntöinä toimivat arviointilistat, arkkitehtuurikatselmoinnit, auditointi, dokumentaatio, käyttäjäpolkujen tarkastelu ja arviointityöpajat, jotka puolestaan auttavat tunnistamaan vaikutuksia, joita ei saada näkyviin pelkillä teknisillä mittareilla.

Kuudes lähtökohta on vastuunjako periaatteiden selkeyttäminen. Haastatteluaineistossa todettiin, että vastuu kestävyydestä ovat usein epäselvä ja jää yksittäisten henkilöiden arvioinnin varaan. Organisaatiosta löytyy vastuullisuusrooleja sekä raportointikäytäntöjä, mutta ohjelmistoprojekteissa ei ole selvää, kuka vastaa kestävyyden huomioimisesta eri osa alueilla. Vastuu jää usein yksittäiselle kehittäjälle tai arkkitehdille, vaikka päätösvalta kestävyyden huomioimisesta olisi asiakkaalla, tuoteomistajalla, projektipäälliköllä tai organisaation johdolla. Kestävyyden huomioiminen edellyttää näkyväksi tehtyä vastuunjakoa.

Muodostettavan mallin tarkoituksena on koota näistä lähtökohdista käytännöllinen ja toimiva kokonaisuus. Malli yhdistää kestävyyden ulottuvuudet, ohjelmistojen vaikutustasot, ohjelmistokehityksen elinkaarivaiheet, arviointi- ja mittauskäytännöt sekä vastuunjaon. Huomioimalla kaikki nämä osa-alueet, kestävyys voidaan mallintaa konkreettisiksi kysymyksiksi, tehtäviksi ja päätöksiksi ohjelmistokehityksen eri vaiheisiin. Mallin

tarkoituksena ei ole korvata olemassa olevia ohjelmistokehitysmenetelmiä, vaan tarjota niitä täydentävä tarkastelumalli, jonka avulla kestävyyttä voidaan käsitellä järjestelmällisesti osana ohjelmiston elinkaarta. Malli muodostaa sillan teoreettisen viitekehyksen ja käytännön ohjelmistokehityksen välille. Se kokoaa yhteen tutkimuksessa tunnistetut keskeiset käsitteet ja empiiriset havainnot sekä jäsentää ne malliksi, jota voidaan hyödyntää ohjelmistoprojekteissa.

5.2 Mallin perusrakenne

Kestävyysmallin perusrakenne muodostuu viidestä osa-alueesta, jotka jokainen täydentävät toisiaan. Nämä ovat kestävyyden ulottuvuudet, ohjelmistojen vaikutustasot, ohjelmistokehityksen elinkaarivaiheet, arviointi- ja mittauskäytännöt sekä vastuunjako. Mallin tarkoituksena on yhdistää nämä osa-alueet niin, että kestävyyttä voidaan tarkastella ohjelmistokehityksessä järjestelmällisesti. Mallin rakenne perustuu ajatukseen, että ohjelmiston kestävyys ei synny yhdestä päätöksestä tai yksittäisestä kehitysvaiheesta, vaan se muodostuu useiden ratkaisujen yhteisvaikutuksena.

Ensimmäinen osa-alue on kestävyyden ulottuvuuksien määrittely. Kestävyys ymmärretään mallissa moniulotteisesti, johon sisältyvät ympäristöllinen, sosiaalinen, taloudellinen, tekninen ja yksilöllinen kestävyys. Tämä on huomionarvoista, koska haastatteluaineiston perusteella kestävyys käytännössä rajoittui tekniseen pitkäikäisyyteen, kustannustehokkuuteen tai resurssien käyttöön. Kaikki edellä mainitut ovat tärkeitä kestävyyden näkökulmia, mutta ne eivät kuvaa yksinään ohjelmiston kestävyyttä.

Toinen osa-alue on vaikutustasojen erottaminen. Mallissa ohjelmistojen vaikutukset jaetaan suoriin, epäsuoriin ja systeemisiin vaikutuksiin. Suorat vaikutukset liittyvät ohjelmiston omaan toimintaan ja infrastruktuuriin, kuten energiankulutukseen, laskentaan, tallennukseen, datansiirtoon ja palvelinten käyttöön. Epäsuorat vaikutukset syntyvät siitä, miten ohjelmisto muuttaa prosesseja, työnkulkuja, käyttäjän toimintaa tai organisaation toimintatapoja. Systeemiset vaikutukset ovat laajempia ja voivat liittyä esimerkiksi palvelurakenteiden, asiointitapojen, työprosessien tai käyttäytymisen muutoksiin. Keskeistä mallissa on vaikutustasojen erottaminen. Ohjelmiston kestävyys ei näy vain sen omassa energiankulutuksessa. Toisaalta ohjelmisto voi kuluttaa jonkin verran resursseja, mutta vähentää kokonaiskuormaa, jos se vähentää tarpeettomia työvaiheita tai parantaa palveluprosessia.

Kolmas osa-alue on ohjelmistokehityksen elinkaarivaiheet. Mallissa kestävyys kytketään koko ohjelmiston elinkaareen projektin alkuvaiheesta käytöstä poistoon asti. Tällä tarkoitetaan sitä, että kestävyyttä ei arvioida ainoastaan toteutuksessa tai käytön aikana, vaan se huomioidaan siinä vaiheessa, kun päätetään, mitä ollaan rakentamassa ja miksi. Elinkaarinäkökulma jakaa mallin käytännön soveltamisen vaiheisiin. Näitä ovat projektin alkuvaihe, vaatimusten määrittely, suunnittelu ja arkkitehtuuri, toteutus, testaus, käyttöönotto, käyttö ja ylläpito sekä käytöstä poisto. Jokaisessa vaiheessa kestävyyttä tarkastellaan eri tavalla. Projektin alkuvaiheessa kysymys liittyy siihen, rakennetaanko oikea järjestelmää ja rakennetaanko se oikein. Arkkitehtuurissa on huomioitava ohjelmiston pitkäikäisyys, muunneltavuus ja teknologiset valinnat. Toteutuksessa ja testauksessa painotetaan resurssitehokkuutta, laatua ja suorituskykyä. Ylläpidossa ja käytöstä poistossa keskitytään teknisen velan hallintaan, tarpeettomien toimintojen poistamiseen ja vanhojen järjestelmien hallittuun alasajoon.

Neljäs osa-alue on arviointi- ja mittauskäytännöt. Ohjelmiston vaikutukset ovat monitasoisia, joten mallissa ei pyritä arvioimaan kestävyyttä vain yhdellä mittarilla. Osa vaikutuksista on määrällisiä ja osa edellyttää laadullista arviointia. Mittarit tekevät ohjelmiston suoria vaikutuksia näkyväksi, mutta ne eivät riitä kertomaan kaikkea vaikutuksista käyttäjiin, sosiaalisista vaikutuksista, prosessien muutoksista tai systeemisistä seurauksista. Koska nämä eivät riitä, malliin tarvitaan myös laadullisia arviointikäytäntöjä. Laadullinen arviointi auttaa tunnistamaan sellaisia vaikutuksia, jotka eivät näy suoraan teknisissä mittareissa mutta ovat kestävyyden kannalta olennaisia.

Viidentenä osa-alueena mallissa on vastuunjako. Mallissa kestävyys nähdään jaettuna vastuuna, mutta ei epämääräisenä yhteisvastuuna. Tämä tarkoittaa, että eri rooleilla on erilaisia tehtäviä kestävyyden huomioimisessa. Tavoitteisiin, priorisointiin ja resursointiin vaikuttaa pääsääntöisesti tilaaja tai asiakas. Vaatimukseksi tai osaksi projektin ohjausta, kestävyyden voi määritellä tuoteomistaja ja projektipäällikkö. Arkkitehdillä on mahdollisuus vaikuttaa teknisiin ratkaisuihin, elinkaareen ja infrastruktuuriin. Kehittäjät ja testaajat määrittelevät toteutuksen laadun, ylläpidettävyyden, resurssitehokkuuden ja testauksen laadun. Organisaation vastuullisuustoimijat voivat tukea mittaamista, raportointia ja arviointikäytäntöjä. Loppukäyttäjien rooli näkyy käytönaikaisissa vaikutuksissa, palautteessa ja siinä, miten ohjelmisto muuttaa käytännössä arjen toimintaa. Vastuunjaon tavoitteena on estää sellaiset tilanteet, jossa kestävyys jää yksittäisen asiantuntijan oma-aloitteisuuden varaan. Kestävyyden huomioiminen vaatii päätöksentekoa, resursointia ja seurantaa. Siksi

vastuuta pitää tarkastella suhteessa siihen, kenellä on mahdollisuus vaikuttaa projektin tavoitteisiin, teknisiin valintoihin, budjettiin, aikatauluun ja ylläpitoon. Mallin perusrakenne voidaan tiivistää seuraavasti.

Osa-alue	Tarkoitus	Esimerkki käytännön tarkastelusta
Kestävyyden ulottuvuudet	Jäsentää, mitä kestävyyden osa-alueita ohjelmistossa tarkastellaan	Ympäristöllinen, sosiaalinen, taloudellinen, tekninen ja yksilöllinen kestävyys
Vaikutustasot	Erottaa, millaisia vaikutuksia ohjelmisto voi synnyttää	Suorat, epäsuorat ja systeemiset vaikutukset
Elinkaarivaiheet	Sijoittaa kestävyyden arviointi ohjelmistokehityksen eri vaiheisiin	Määrittely, arkkitehtuuri, toteutus, testaus, käyttö, ylläpito ja käytöstä poisto
Arviointi- ja mittauskäytännöt	Tekee vaikutuksista tunnistettavia ja seurattavia	Mittarit, checklistit, dokumentaatio, auditointi ja arviointityöpajat
Vastuunjako	Määrittää, kenen tehtävä on huomioida ja seurata kestävyyttä	Asiakas, toimittaja, tuoteomistaja, projektipäällikkö, arkkitehti, kehittäjä ja loppukäyttäjä

Taulukko 6. Kestävyysmallin keskeiset osa-alueet

Taulukossa 6 on esitetty toteutettavan mallin perusajatukset. Kestävyyttä ei ole mallissa vain abstraktina tavoitteena, vaan se on jaettu osa-alueisiin, joiden kautta kestävyys tuodaan osaksi ohjelmistokehityksen käytäntöjä. Malli auttaa jokaista sidosryhmää kysymään, mitä kestävyyden ulottuvuutta tarkastellaan, millaisesta vaikutuksesta on kyse, missä elinkaaren vaiheessa vaikutus syntyy, miten tätä vaikutusta voidaan arvioida ja kuka siitä vastaa. Tämä mallin perusrakenne toimii runkona seuraavalle alaluvulle, jossa mallia sovelletaan ohjelmistokehityksen elinkaaren vaiheisiin ja mallin eri osa-alueet muutetaan käytännön kysymyksiksi, toimenpiteiksi, arviointitavoiksi sekä vastuunjaoksi.

5.3 Mallin soveltaminen ohjelmiston elinkaaressa

Kestävyysmallin käytännön soveltaminen tapahtuu ohjelmistokehityksen elinkaaren eri vaiheissa. Mallin soveltaminen perustuu siihen, että jokaisessa elinkaarivaiheessa kysytään viisi asiaa: mitä kestävyyden ulottuvuutta tarkastellaan, millaisia vaikutuksia voi syntyä, miten vaikutuksia voidaan arvioida, mitä toimenpiteitä tarvitaan ja kuka on vastuussa niiden huomioimisesta. Taulukossa 7 esitetään malli käytännön soveltamiseen ohjelmistokehityksen elinkaaren vaiheissa.

Elinkaarivaihe	Kestävyyskysymys	Toimenpide	Arviointitapa	Päävastuu
Projektin alkuvaihe	Rakennetaanko oikea järjestelmä? Mitä kestävyyshyötyjä syntyy? Mitä kestävyyshaittoja syntyy?	Tunnistetaan järjestelmän tarkoitus, käyttäjät, prosessit, vaikutukset ja alustavat kestävyystavoitteet.	Vaikutusarvio Sidosryhmäanalyysi Prosessien kuvaus Voice of Customer Alustava kestävyyden arviointi	Asiakas Tuoteomistaja Projektipäällikkö Ohjelmiston toimittaja
Vaatimusmäärittely	Onko kestävyys kirjattu vaatimuksiin ja hyväksymiskriteereihin?	Kestävyystavoitteet muunnetaan ei-toiminnallisiksi vaatimuksiksi, arkkitehtuuriperiaatteiksi ja backlog-tehtäviksi.	NFR-listaus Hyväksymiskriteerit Definition of Done Vaatimuskatselmointi Riskikatselmointi	Tuoteomistaja Asiakas Arkkitehti Projektipäällikkö
Suunnittelu ja arkkitehtuuri	Tukeeko ratkaisu pitkäikäisyyttä, ylläpidettävyyttä ja resurssitehokkuutta?	Tarkoituksenmukaiset arkkitehtuuri valinnat Teknologiat Ajopaikka Skaalautuvuusratkaisut Integraatiot	Arkkitehtuurikatselmointi Teknologiavalintojen arviointi Kapasiteettikustannusarvio Pilvikustannusarvio Elinkaaririskien arviointi	Arkkitehti Tekninen johto Kehitystiimi Asiakas
Toteutus	Toteutetaanko ohjelmisto ylläpidettävästi ja tehokkaasti?	Vältetään tarpeetonta koodia, raskaita riippuvuuksia ja ylimitoitettuja ratkaisuja. Huolehditaan koodin laadusta ja ylläpidettävyydestä.	Code review Staattinen analyysi Resurssien käyttö CI/CD-seuranta Teknisen velan seuranta	Kehittäjät Arkkitehti Tekninen vetäjä
Testaus	Testataanko myös kestävyyttä tukevia ominaisuuksia?	Testataan suorituskykyä, kuormitusta, saavutettavuutta, skaalautuvuutta ja resurssien käyttöä.	Kuormitustestit Vasteajat SLA-rajat Pilvimittarit Saavutettavuustestit Energian/resurssien käytön arviot	Testaajat Kehittäjät Arkkitehti Tuoteomistaja
Käyttöönotto	Miten järjestelmä muuttaa käyttäjien toimintaa ja organisaation prosesseja?	Käyttäjien tukeminen muutoksessa, käyttöönoton vaikutusten seuranta. Varmistetaan, ettei uusi järjestelmä lisää tarpeetonta työtä.	Käyttäjäpalaute Käyttöönoton seuranta Tukipyyntöjen analyysi Käyttäjäpolkujen tarkastelu	Projektipäällikkö Asiakas Tuoteomistaja Käyttäjätuki
Käyttö ja ylläpito	Säilyykö järjestelmä hyödyllisenä, tehokkaana ja ylläpidettävänä käytön aikana?	Seurataan: käyttöä, kustannuksia, resurssien kulutusta, teknistä velkaa ja tarpeettomia toiminnallisuuksia.	Pilvikustannukset CPU- ja muistinkäyttö CO_2-arviot Käyttöaste Backlog Auditointi Ylläpitokatselmointi	Ylläpito Asiakas Toimittaja Arkkitehti Vastuullisuushenkilöt
Käytöstä poisto	Poistetaanko tarpeettomat järjestelmät, resurssit ja data hallitusti?	Suunnitellaan alasajo Poistetaan tarpeettomat ympäristöt Arkistoidaan tai siirretään data Puretaan integraatiot	Alasajosuunnitelma Dokumentaatio Resurssien vapautuksen seuranta Datan hallinnan tarkistus	Asiakas Ylläpito Arkkitehti Tietohallinto Ohjelmiston toimittaja

Taulukko 7. Mallin soveltaminen ohjelmiston elinkaaressa

Taulukko 7 osoittaa, että kestävyyden huomioiminen alkaa jo ennen varsinaista toteutusta ja jatkuu ohjelmiston käyttöön, ylläpitoon ja käytöstä poistoon niin, että elinkaaren eri vaiheissa kestävyydellä on eri painotus.

Projektin alkuvaiheessa on varmistettava, että ratkaistavaa ongelmaa ei määritellä liian kapeasti. Alkuvaiheessa arvioidaan, rakennetaanko oikeaa järjestelmää, mitä toimintaa

järjestelmä muuttaa ja millaisia myönteisiä tai kielteisiä vaikutuksia ratkaisulla siitä muodostuu eri sidosryhmille. Mikäli järjestelmän tavoite tai rajaus määritellään liian suppeasti, myöhemmissä vaiheissa usein voidaan enää vaikuttaa tekniseen toteutukseen, mutta ei siihen, onko ratkaisu kokonaisuuden kannalta tarkoituksenmukainen.

Vaatimusmäärittelyssä kestävyys muutetaan yleisestä tavoitteesta konkreettisiksi vaatimuksiksi. Ilman tätä vaihetta kestävyys jää helposti yleiseksi periaatteeksi ja myös kestävyyden vastuuhenkilö puuttuu projektista.

Suunnittelu- ja arkkitehtuurivaiheessa korostuvat pitkäikäisyys, muunneltavuus ja resurssitehokkuus. Arkkitehtuuri- ja teknologia valinnat sekä integraatiot ja kapasiteettiratkaisut vaikuttavat siihen, kuinka paljon ohjelmisto kuluttaa resursseja, kuinka helposti sitä voidaan ylläpitää ja miten hyvin se muokattavissa tulevaisuuden tarpeisiin. Tässä vaiheessa tehtävät päätökset vaikuttavat siis koko ohjelmiston elinkaareen.

Toteutus- ja testausvaiheessa kestävyys näkyy teknisenä laatuna. Selkeä, tehokas ja ylläpidettävä koodi vähentää teknistä velkaa ja voi samalla pienentää resurssien käyttöä. Testauksessa kestävyyttä voidaan tarkastella suorituskyvyn, skaalautuvuuden, kuormituksen, saavutettavuuden ja palvelutasojen kautta. Vaikka ympäristökestävyyden suora mittaaminen voi olla vaikeaa, usein tekniset mittarit toimivat sen epäsuorina indikaattoreina.

Käyttöönotossa ja käytössä mallin painopiste siirtyy ohjelmistoprojektista käyttäjiin ja todellisiin muutoksiin prosesseissa. Teknisesti onnistunut ohjelmisto voi olla kestävyyden kannalta ongelmallinen, jos se monimutkaistaa prosesseja tai kuormitus siirtyy toiseen kohtaan. Tämän takia käytönaikainen seuranta, käyttäjäpalaute ja käyttäjäpolkujen tarkastelu ovat tärkeä osa muodostettavaa mallia.

Ohjelmisto muuttua ajan myötä kestämättömäksi, jos sitä ei ylläpidetä aktiivisesti ja siihen jää vanhaa koodia tai tarpeettomia ominaisuuksia sekä käyttämättömiä ympäristöjä. Käytöstä poiston suunnittelu on osa kestävyyttä, koska tarpeettomasti käytössä olevat ympäristöt kuluttavat resursseja ilman vastaavaa hyötyä. Kokonaisuutena malli ohjaa tarkastelemaan ohjelmistokehitystä elinkaaren kaikissa vaiheissa, jossa jokaisella vaiheella on oma roolinsa kestävyyden toteutumisessa. Mallin tekee kestävyyden näkyväksi ja havaitaan, mitä vaikutuksia ohjelmistolla on, missä vaiheessa niihin voidaan vaikuttaa, miten niitä voidaan arvioida ja kuka vastaa niiden huomioimisesta.

5.4 Vastuunjako kestävyysmallissa

Kestävyysmallin toimivuus edellyttää, että vastuu kestävyyden huomioimisesta on jaettu oikeille tahoille ja samalla tehty konkreettiseksi. Haastatteluaineiston perusteella kestävyys jää helposti yksittäisten kehittäjien, arkkitehtien tai vastuullisuudesta kiinnostuneiden asiantuntijoiden varaan, mikäli sille ei ole määritelty selkeää paikkaa projektin päätöksenteossa. Yksittäisellä asiantuntijalla ei välttämättä ole mahdollisuutta vaikuttaa projektin tavoitteisiin, budjettiin, aikatauluun tai asiakkaan prioriteetteihin, jolloin tästä syntyy kestävyys ongelma.

Mallissa vastuunjako perustuu siihen, että eri sidosryhmillä on mahdollisuus vaikuttaa kestävyyteen erilaisissa kohdissa ohjelmiston elinkaarta. Tilaavalla asiakkaalla on keskeinen rooli tavoitteiden, vaatimusten ja resursoinnin määrittelyssä. Ohjelmiston toimittaja ja kehitystiimi vaikuttavat siihen, millaisia teknisiä ratkaisuja toteutukseen ehdotetaan ja toteutetaan. Arkkitehti, kehittäjät ja testaajat vastaavat lopulta käytännön teknisistä ratkaisuista, mutta heidän työnsä tarvitsee tuekseen asiakkaan määrittämät projektin tavoitteet, hyväksymiskriteerit ja päätöksenteon rakenteet. Vastuu kestävyydestä ei siis ole vain tekninen kysymys.

Kestävyysmallissa vastuunjako rakentuu asiakkaan, toimittajan, projektin avainroolien, organisaation vastuullisuustiimien ja loppukäyttäjien välille. Tämä vastaa tutkimuksen teoreettista lähtökohtaa, jossa ohjelmiston vaikutukset syntyvät sekä teknisistä ratkaisuista että siitä, miten ohjelmistoa käytetään osana laajempaa sosio-teknistä toimintaympäristöä. Vastuunjako liittyy myös SRoSE-ajatteluun, jossa vastuu ohjelmiston päästöistä ja niiden vaikutuksista jakautuvat hankkijan, ohjelmistontoimittajan ja loppukäyttäjän välille.

Toimija	Vastuu kestävyysmallissa	Käytännön tehtävät
Asiakas / tilaaja	•tavoitteiden määrittely •priorisointi •resursointi •hyväksymiskriteerien	•sisällyttää kestävyystavoitteet hankintaan •määrittää kestävyyteen liittyvät vaatimukset •varaa aikaa ja budjettia kestävyyden huomioimiseen
Tuoteomistaja	•kestävyyden vieminen tuotteen kehitykseen •backlog-priorisointi •käyttäjäarvon ja	•muuttaa kestävyystavoitteet backlog-tehtäviksi •lisää kestävyyteen liittyviä hyväksymiskriteerejä •huomioi käyttäjävaikutukset ja prosessimuutokset
Projektipäällikkö	•kestävyyden huomioiminen projektin ohjauksessa •aikataulun, vastuiden ja päätösten hallinta	•aikatauluttaa arvioinnit ja katselmoinnit •varmistaa, että vastuut on määritelty •nostaa kestävyyteen liittyvät riskit päätöksentekoon
Arkkitehti **Tekninen johto**	•teknisten ratkaisujen pitkäikäisyys •arkkitehtuurin ylläpidettävyys •infrastruktuurin ja ajopaikan	•arvioi teknologiavalinnat •suunnittelee skaalautuvuuden ja integraatiot •huomioi pilviympäristön, kapasiteetin ja ajopaikan
Kehittäjät	•ylläpidettävä ja resurssitehokas toteutus •koodin laatu •teknisen velan hallinta	•kirjoittavat selkeää ja ylläpidettävää koodia •välttävät tarpeetonta monimutkaisuutta •osallistuvat code review -käytäntöihin •seuraavat riippuvuuksia ja teknistä velkaa
Testaajat **Laadunvarmistus**	•kestävyyttä tukevien ominaisuuksien näkyväksi tekeminen •suorituskyvyn ja käytettävyyden	•testaavat suorituskykyä ja kuormitusta •seuraavat vasteaikoja ja SLA-rajoja •testaavat saavutettavuutta ja skaalautuvuutta •tuovat testitulokset päätöksenteon tueksi
Ylläpito **Operointi**	•käytönaikaisten vaikutusten seuranta •resurssien hallinta- teknisen velan ja tarpeettomien toimintojen	•seuraa käyttöastetta, kustannuksia ja pilviresursseja •valvoo hälytyksiä ja palvelutasoa •tunnistaa tarpeettomat resurssit ja ominaisuudet
Organisaation **vastuullisuushenkilöt**	•kestävyyden mittaamisen ja raportoinnin tuki •ohjeistusten ja arviointikäytäntöjen kehittäminen •organisaatiotason	•tuottavat ohjeita ja arviointilistoja •tukevat mittareiden ja raportoinnin määrittelyä •osallistuvat auditointeihin ja arviointikäytäntöihin •auttavat yhdistämään projektitason havainnot
Loppukäyttäjät	•käytön aikaisten vaikutusten näkyväksi tekeminen •käyttäjäpolkujen ja työn muutosten esiin tuominen •palautteen antaminen	•antavat palautetta järjestelmän käytöstä •osoittavat kohdat, joissa työ helpottuu tai vaikeutuu •tuovat esiin tarpeettomat vaiheet ja kuormitusta lisäävät kohdat

Taulukko 8. Vastuunjako kestävyysmallissa

Taulukko 8 osoittaa, että kestävyyden huomioiminen ei ole yhden yksittäisen henkilöroolin vastuulla. Asiakas voi määrittää kestävyyden tavoitteet ja maksaa niiden toteuttamisesta, mutta toimittajan vastuulla on tuoda esiin tekniset vaihtoehdot ja niiden kestävyysvaikutukset. Projektipäällikön ja tuoteomistajan tehtävänä on varmistaa, että kestävyys ei jää yleiseksi periaatteeksi, vaan siirtyy projektin ohjaukseen, backlogille ja hyväksymiskriteereihin.

Arkkitehti ja kehitystiimi omalta osaltaan tekevät ratkaisuja, joilla on vaikutuksia ohjelmiston tekniseen kestävyyteen koko sen elinkaaren ajan. Ylläpidon rooli on yhtä tärkeä, koska monet kestävyyteen liittyvät vaikutukset alkavat näkyvät vasta käytön aikana. Käyttöaste, pilviresurssit, kustannukset, tekninen velka, tarpeettomat ominaisuudet ja vanhentuneet riippuvuudet eivät ole havaittavissa vielä projektin alkuvaiheessa. Siksi mallissa vastuu ei pääty käyttöönottoon, vaan jatkuu myös ylläpidossa ja käytöstä poistossa.

Organisaation vastuullisuushenkilöiden tehtävä ei ole korvata projektissa sidosryhmien omaa vastuuta, vaan olla tukemassa sitä. Vastuullisuushenkilöillä on mahdollisuus tarjota mittareita, ohjeita, arviointilistoja, auditointikäytäntöjä ja raportointiin liittyvää laajaa osaamista. Näin vastuullisuustyö kytketään paremmin osaksi ohjelmistokehityksen käytäntöjä.

Loppukäyttäjien rooli liittyy erityisesti ohjelmiston epäsuoriin ja systeemisiin vaikutuksiin. Käyttäjäpalauteen, käyttötapojen ja käyttäjäpolkujen seurantaa osoittaa, onko ohjelmisto vähentänyt tehtävää työtä tai vähentänyt hukkaa vai onko se lisännyt kuormitusta. Tästä syystä käyttäjävaikutusten arviointi on osa kestävyyden vastuunjakoa, vaikka loppukäyttäjä ei vastaa ohjelmiston suunnittelusta tai toteutuksesta. Kokonaisuutena vastuunjaon tarkoituksena on estää tilanne, jossa kestävyys jää ilman omistajuutta ja se muuttuu yhteisvastuulliseksi. Mallissa kaikilla rooleilla on oma vaikutuskohtansa: asiakas määrittää tavoitteet, projektin roolit vievät ne käytäntöön, tekniset asiantuntijat toteuttavat ja arvioivat ratkaisut, ylläpito seuraa käytönaikaisia vaikutuksia ja organisaation vastuullisuustoimijat tukevat mittaamista ja raportointia. Näin kestävyys voidaan siirtää yksittäisten henkilöiden kiinnostuksesta osaksi ohjelmistokehityksen normaalia ohjausta ja päätöksentekoa.

5.5 Kestävyysmallin vertailu aiempiin viitekehyksiin

Viitekehys	Sisältö	Kestävyysmalli
GREENSOFT / GSLC	Kestävyyden tarkastelu ohjelmiston elinkaaressa	Täydentää elinkaarinäkökulmaa vaatimustenhallinnan, vastuunjaon, mittareiden, kädenjäljen ja sovellettavuuden näkökulmilla.
SusAF	Kestävyyden tunnistaminen moniulotteisena kysymysten kautta	Siirtää tunnistamisen projektin ohjaukseen, backlogille, hyväksymiskriteereihin ja ylläpitoon.
Karlskrona Manifesto	Periaatteellinen pohja kestävälle ohjelmistosuunnittelulle	Periaatteen konkreettinen soveltaminen ohjelmistokehityksen vaiheisiin ja vastuun jako
ICT vaikutustasot ***(Hilty & Aebischer, 2015)***	Suorat, epäsuorat ja systeemiset vaikutukset	Vaikutustasojen liittäminen konkreettisesti ohjelmistokehityksen päätöksiin
Green IT / Green by IT	Erotetaan hiilijalanjälki ja hiilikädenjälki	Yhdistää jalanjäljen, kädenjäljen, haitalliset vaikutukset ja rebound-riskit
Kestävä projektinhallinta	Projektin alkuvaiheen määrittely, sidosryhmät, vastuunjako	Soveltaa projektinhallinnan näkökulmaa ohjelmistokehityksen elinkaareen

Taulukko 9. Kestävyysmallin vertailu aiempiin viitekehyksiin

Taulukko 9 osoittaa, että tässä työssä muodostettu kestävyysmalli ei korvaa aiempia viitekehyksiä, vaan kokoaa niiden keskeiset näkökulmat käytännön ohjelmistokehityksen kontekstiin. Aiemmat mallit painottavat hieman eri näkökulmia. GSLC-malli keskittyy ohjelmiston elinkaareen, SusAF kestävyyden moniulotteiseen tunnistamiseen ja Karlskrona Manifesto kestävän ohjelmistosuunnittelun periaatteisiin. ICT:n vaikutustasot sekä Green IT / Green by IT -jaottelu täydentävät kokonaisuutta, koska ne auttavat erottamaan ohjelmistojen suorat, epäsuorat, systeemiset ja kädenjälkeen liittyvät vaikutukset toisistaan. Tässä työssä kehitetty malli yhdistää nämä näkökulmat ohjelmistokehityksen eri vaiheisiin, vastuunjakoon, arviointikäytäntöihin ja elinkaaren aikaiseen päätöksentekoon.

6 Johtopäätökset ja jatkotutkimus

Tässä luvussa esitetään tutkimuksen keskeiset johtopäätökset ja arvioidaan tuloksia suhteessa tutkimuskysymyksiin. Lisäksi luvussa tarkastellaan tutkimuksen rajoituksia ja esitetään jatkotutkimusaiheita. Tutkimuksen tavoitteena oli selvittää, miten kestävän kehityksen ja kädenjälkivaikutusten näkökulmat voidaan integroida ohjelmistojen suunnittelu- ja kehitysprosesseihin koko ohjelmiston elinkaaren aikana. Työssä tarkasteltiin kestävyyttä ohjelmistokehityksen laatuattribuuttina, ei-toiminnallisena vaatimuksena sekä suorien, epäsuorien ja systeemisten vaikutusten näkökulmasta.

Tutkimus vastaa johdannossa tunnistettuun tutkimusaukkoon kokoamalla kestävän ohjelmistokehityksen hajanaisia teoreettisia näkökulmia yhdeksi elinkaarilähtöiseksi tarkasteluksi ja täydentämällä niitä empiirisellä haastatteluaineistolla. Työssä osoitetaan, että kestävyys tunnistetaan ohjelmistokehityksessä yhä useammin tärkeäksi näkökulmaksi, mutta sen käsittely ei vielä ole systemaattisesti vakiintunut vaatimustenhallintaan, arkkitehtuuripäätöksiin, mittaamiseen tai vastuunjakoon. Tutkimuksen muodostama kestävyysmalli vastaa tähän puutteeseen kokoamalla viitekehyksen, jossa kestävyyden ulottuvuuksia, ohjelmiston elinkaarivaiheita, vaikutustasoja, arviointikäytäntöjä ja vastuita voidaan tarkastella samanaikaisesti. Malli ei kuitenkaan vielä ole valmis määrällinen mittaristo, vaan laadullista arviointia ja päätöksentekoa tukeva viitekehys, jonka jatkokehittäminen edellyttää soveltamista todellisissa ohjelmistoprojekteissa.

Tutkimuksen perusteella voidaan todeta, että kestävyys tunnistetaan ohjelmistokehityksessä aiempaa paremmin, mutta sen käytännön asema on vielä epäyhtenäinen. Kestävyys näkyy erityisesti teknisen pitkäikäisyyden, ylläpidettävyyden, resurssitehokkuuden, kustannusten, sosiaalisen vastuun ja eettisten vaikutusten kautta. Se ei kuitenkaan useimmiten ole selkeä vaatimus, hyväksymiskriteeri, mittari tai projektin ohjauskohde.

Keskeinen johtopäätös on, että kestävyyden integrointi ohjelmistokehitykseen edellyttää siirtymistä yleisestä vastuullisuuspuheesta konkreettisiin käytäntöihin. Pelkkä tekninen optimointi tai energiatehokkuuden tarkastelu ei riitä, koska ohjelmistojen merkittävät vaikutukset syntyvät usein käytön, prosessimuutosten ja laajempien systeemisten muutosten kautta. Kestävyys tulee kytkeä vaatimustenhallintaan, arkkitehtuuripäätöksiin, toteutukseen, testaukseen, ylläpitoon ja käytöstä poistoon. Ohjelmisto voi vähentää hukkaa ja parantaa työn

sujuvuutta, mutta se voi myös lisätä käyttäjän työtä, kasvattaa kokonaiskäyttöä, synnyttää teknistä velkaa tai ylläpitää tarpeettomia järjestelmiä.

Toinen johtopäätös on, että ohjelmiston kestävyyttä on arvioitava koko elinkaaren näkökulmasta. Tärkeimmät vaikutusmahdollisuudet sijoittuvat ohjelmistoprojektin alkuvaiheeseen, vaatimusten määrittelyyn ja suunnitteluun sekä arkkitehtuuriin. Näissä vaiheissa päätetään, mitä järjestelmää rakennetaan, miksi se rakennetaan ja millaiset tekniset ratkaisut ohjaavat sen käyttöä pitkällä aikavälillä. Kestävyyden kannalta myös ylläpito ja käytöstä poisto ovat tärkeitä. Ohjelmisto voi muuttua kestämättömäksi, jos siihen kertyy teknistä velkaa, tarpeettomia toiminnallisuuksia, vanhentuneita riippuvuuksia tai käyttämättömiä resursseja. Siksi kestävyyden arviointi ei voi päättyä käyttöönottoon.

Kolmas johtopäätös liittyy mittaamiseen. Organisaatioissa käytetään jo mittareita, jotka voivat tukea kestävyyden arviointia epäsuorasti. Tällaisia ovat esimerkiksi saatavilla olevat pilvikustannukset, CPU-tasot, vasteajat, SLA-rajat, käyttöaste ja teknisen velan seuranta. Ne eivät kuitenkaan vielä muodosta yhtenäistä kestävyyden mittaristoa. Kestävyyden arviointi edellyttää siis määrällisten mittareiden ja laadullisten arviointikäytäntöjen yhdistämistä. Teknisten mittareiden rinnalle tarvitaan arkkitehtuurikatselmointeja, käyttäjäpolkujen tarkastelua, dokumentaatiota, auditointia ja arviointityöpajoja.

Neljäs johtopäätös koskee vastuita. Kestävyys ei voi jäädä yksittäisen kehittäjän tai arkkitehdin vastuulle, koska siihen liittyvät päätökset koskevat myös tavoitteita, budjettia, aikataulua, priorisointia, teknologiaa, ylläpitoa ja käyttäjien toimintaa. Vastuu jakautuu asiakkaan, toimittajan, projektipäällikön, tuoteomistajan, arkkitehdin, kehittäjien, testaajien, ylläpidon, vastuullisuustiimien ja loppukäyttäjien välille.

Tutkimuksen kokonaisjohtopäätöksenä voidaan todeta, että kestävä ohjelmistokehitys edellyttää teknisen, organisaation sisäisen ja systeemisen tarkastelun yhdistämistä. Kestävyys ei ole vain energiankulutukseen liittyvä kysymys, vaan se liittyy siihen, millaisia järjestelmiä rakennetaan, miten ne muuttavat toimintaa, kuinka pitkäikäisiä ne ovat ja millaisia vaikutuksia ne tuottavat käyttäjille, organisaatioille ja ympäröivälle yhteiskunnalle.

6.1 TK1: Miten kestävän kehityksen ja kädenjälkivaikutusten näkökulmat voidaan integroida ohjelmistokehityksen eri vaiheisiin?

Kirjallisuuden perusteella kestävyys tulee integroida ohjelmistokehitykseen koko elinkaaren läpäisevänä näkökulmana. Tämä tarkoittaa, että kestävyyttä ei käsitellä vain toteutusvaiheen

teknisenä optimointina, vaan se huomioidaan jo projektin alkuvaiheessa, vaatimusten määrittelyssä, suunnittelussa, arkkitehtuurissa, toteutuksessa, testauksessa, käyttöönotossa, käytössä, ylläpidossa ja käytöstä poistossa. Haastatteluissa kävi ilmi, että kestävyys ohjelmistoalalla toteutetaan yksittäisinä, erillisinä toimina, mutta ei kokonaisvaltaisena kestävyysajatteluna. Ohjelmistotaloissa ei haastattelujen perusteella ole kolmeen pilariin perustuvaa kulttuuria, vaan kestävyys on sirpaloitunutta yksittäisten työntekijöiden tietämykseen ja mielenkiintoon perustuvaa tekemistä.

Keskeistä on, että kestävyys muutetaan yleisestä tavoitteesta konkreettisiksi vaatimuksiksi, päätöksiksi ja arviointikäytännöiksi. Projektin alkuvaiheessa tämä tarkoittaa sen arvioimista, rakennetaanko oikeaa järjestelmää ja millaisia vaikutuksia järjestelmällä tavoitellaan. Vaatimusmäärittelyssä kestävyyteen liittyvät tavoitteet tulee kirjata ei-toiminnallisiksi vaatimuksiksi, hyväksymiskriteereiksi tai backlog-tehtäviksi. Arkkitehtuurissa ja suunnittelussa huomio kohdistuu pitkäikäisyyteen, ylläpidettävyyteen, ajopaikkaan, teknologioihin ja resurssitehokkuuteen. Toteutuksessa ja testauksessa kestävyys näkyy teknisen laadun, suorituskyvyn, skaalautuvuuden ja resurssien käytön kautta. Ylläpidossa ja käytöstä poistossa korostuvat teknisen velan hallinta, tarpeettomien toiminnallisuuksien poistaminen ja vanhojen järjestelmien hallittu alasajo.

6.2 TK2: Millaisia negatiivisia hiilijalanjälki- ja systeemivaikutuksia huonosti suunnitelluilla ohjelmistoilla voi olla?

Tutkimuksen perusteella huonosti suunniteltu ohjelmisto voi lisätä kuormitusta usealla tavalla. Se voi kuluttaa tarpeettomasti laskenta-, tallennus- tai pilviresursseja, kasvattaa ajokustannuksia, lisätä käyttäjän työtä, monimutkaistaa prosesseja ja synnyttää teknistä velkaa. Negatiiviset vaikutukset eivät aina näy ohjelmiston omassa energiankulutuksessa. Ne voivat syntyä myös siitä, miten ohjelmisto muuttaa käyttäjien, organisaatioiden ja palveluprosessien toimintaa. Huonosti suunniteltu järjestelmä voi lisätä klikkauksia, manuaalista työtä, copy-paste-vaiheita tai käyttäjien tarvetta palata palveluun toistuvasti. Se voi myös ylläpitää vanhoja toimintatapoja, kasvattaa kokonaiskäyttöä tai jättää vanhoja järjestelmiä pyörimään uuden rinnalle. Tällöin ohjelmisto voi teknisesti toimia oikein, mutta sen kokonaisvaikutus on kestävyyden kannalta kielteinen.

6.3 TK3: Miten ohjelmistokehittäjät ja organisaatiot voivat käytännössä tunnistaa, mitata ja hallita näitä vaikutuksia?

Tutkimuksen perusteella vaikutusten tunnistaminen edellyttää sekä teknisiä mittareita että laadullisia arviointikäytäntöjä. Pelkät tekniset mittarit eivät riitä, koska ohjelmistojen merkittävät vaikutukset syntyvät usein käyttäjien toiminnan, prosessien ja organisaation käytäntöjen kautta.

Käytännössä tunnistamista voidaan toteuttaa arkkitehtuurikatselmoinneilla, dokumentaatiolla, käyttäjäpolkujen tarkastelulla, auditoinneilla ja arviointityöpajoilla. Määrällisiä mittareita voisivat olla pilvikustannukset, CPU- ja muistinkäyttö, vasteajat, SLA-rajat, kuormitus, käyttöaste, energiankulutusarviot ja CO_2-ekvivalenttiarviot. Näiden avulla voidaan tunnistaa erityisesti ohjelmiston suoria vaikutuksia.

Epäsuorien ja systeemisten vaikutusten hallinta edellyttää lisäksi käyttäjäpalautetta, prosessien kuvaamista ja käytönaikaista seurantaa. Organisaatioiden on tunnistettava, vähentääkö ohjelmisto todella hukkaa ja parantaako se toimintaa vai siirtääkö se kuormitusta käyttäjille, ylläpidolle tai toiseen järjestelmään.

6.4 TK4: Voidaanko kehittää poikkitieteellinen viitekehys tai mittaristo, joka tukee ohjelmistojen kestävyyden ja kädenjälkivaikutusten systemaattista arviointia?

Tämän tutkimuksen perusteella tällainen viitekehys on tarpeellinen. Sen tulee yhdistää useita näkökulmia, sillä yksittäinen mittari ei riitä kuvaamaan ohjelmiston kestävyyttä kokonaisuutena. Tässä työssä muodostettu kestävyysmalli vastaa tähän tarpeeseen. Malli yhdistää kestävyyden ulottuvuudet, ohjelmistojen suorat, epäsuorat ja systeemiset vaikutukset, ohjelmistokehityksen elinkaarivaiheet, arviointi- ja mittauskäytännöt sekä vastuunjaon. Mallin tarkoituksena on tehdä kestävyydestä näkyvä osa ohjelmistokehityksen päätöksentekoa.

Mallia voidaan käyttää erityisesti keskustelun, arvioinnin ja päätöksenteon tukena. Se auttaa tunnistamaan, mitä vaikutuksia ohjelmistolla voi olla, missä vaiheessa niihin voidaan vaikuttaa, miten niitä voidaan arvioida ja kuka vastaa niiden huomioimisesta. Malli ei vielä tarjoa valmista numeerista mittaristoa kaikkiin tilanteisiin, mutta se muodostaa rakenteellisen mallin, jonka pohjalta organisaatiot voivat kehittää omia mittareitaan ja arviointikäytäntöjään.

Yhteenvetona tutkimuskysymyksiin voidaan vastata, että kestävyyden integrointi ohjelmistokehitykseen edellyttää elinkaariajattelua, vaikutustasojen tunnistamista, vaatimustenhallinnan vahvistamista, mittaamisen ja laadullisen arvioinnin yhdistämistä sekä selkeämpää vastuunjakoa. Kestävyys on tuotava osaksi ohjelmistokehityksen arjen päätöksiä ja käytäntöjä, eikä sitä voida jättää yksittäiseksi lisävaatimukseksi projektin muiden tavoitteiden rinnalle.

6.5 Tutkimuksen rajoitukset

Aineisto koostui viidestä puolistrukturoidusta asiantuntijahaastattelusta, joten tutkimuksen tuloksia ei voida pitää tilastollisesti yleistettävinä koko ohjelmistoalalle. Aineisto tarjoaa kuitenkin laadullisen ja kontekstisidonnaisen näkymän siihen, miten kestävyys ymmärretään ja miten se näkyy ohjelmistokehityksen käytännöissä tutkimukseen osallistuneissa organisaatioissa.

Lisäksi tutkimuksessa tarkasteltiin organisaatioiden käyttämiä mittareita ja arviointikäytäntöjä, mutta työssä ei toteutettu varsinaista määrällistä mittausta ohjelmistojen energiankulutuksesta, hiilijalanjäljestä tai kädenjälkivaikutuksista. Tämän vuoksi tutkimuksen tulokset kuvaavat ensisijaisesti kestävyyden tunnistamista, arviointia ja hallintaa laadullisella tasolla. Tutkimuksessa muodostettu kestävyysmalli on näin ollen käsitteellinen ja käytännön arviointia tukeva malli, eikä sen toimivuutta testattu käytännön ohjelmistoprojekteissa. Malli perustuu teoreettiseen viitekehykseen ja empiirisiin havaintoihin, ja sen jatkokehittäminen edellyttää soveltamista todellisissa ohjelmistoprojekteissa.

6.6 Jatkotutkimus

Jatkotutkimuksessa olisi tärkeää testata tässä työssä muodostettua kestävyysmallia käytännön ohjelmistoprojekteissa. Mallia voitaisiin soveltaa projektin alkuvaiheessa, vaatimusmäärittelyssä, arkkitehtuurikatselmoinnissa tai ylläpidon arvioinnissa. Tämän perusteella voitaisiin arvioida, auttaako malli tunnistamaan kestävyyteen liittyviä vaikutuksia paremmin kuin organisaatioiden nykyiset käytännöt.

Jatkotutkimusta voisi toteuttaa mittareiden kehittämiseen. Tässä diplomityössä kävi ilmi, että ohjelmistokehityksessä on käytössä monia epäsuoria mittareita, kuten pilvikustannuksia, resurssien käyttöä, vasteaikoja ja käyttöastetta. Näitä tulisi jatkossa kehittää niin, että ne

tukevat paremmin ohjelmistojen kestävyyden arviointia. Erityisesti tarvitaan mittareita, jotka yhdistävät tekniset vaikutukset, käyttäjävaikutukset ja elinkaaren aikaiset vaikutukset.

Kolmas jatkotutkimusaihe koskee ohjelmistojen epäsuoria ja systeemisiä vaikutuksia. Näiden vaikutusten tunnistaminen on vaikeaa, koska ne eivät aina näy ohjelmiston teknisissä mittareissa. Havaintojen perusteella voisi tutkia, kuinka paljon organisaatioissa on turhia varalta pyöriviä ohjelmistoympäristöjä, joiden olemassaolon perustetta ei tiedetä.

Yksi tärkeä jatkotutkimusalue on myös tekoälyn vaikutus ohjelmistokehityksen kestävyyteen. Haastatteluaineistossa tekoäly nousi esiin uutena teemana, joka liittyy sekä ohjelmistokehityksen tehostumiseen että kasvavaan laskentatehoon ja sitä kautta energiankulutukseen. Jatkossa tulisi tutkia tarkemmin, miten tekoälytyökalujen käyttö vaikuttaa ohjelmistokehityksen resurssien käyttöön, laatuun, tuottavuuteen ja kestävyyteen.

Teoreettisena jatkotutkimusaiheena olisi tarpeellista vahvistaa kestävyyden asemaa ohjelmistokehityksen laatuattribuuttina ja ei-toiminnallisena vaatimuksena. Nykyisessä tutkimuskirjallisuudessa kestävyys on tunnistettu tärkeäksi ohjelmistokehityksen näkökulmaksi, mutta sen suhde muihin laatuattribuutteihin, kuten tietoturvaan, käytettävyyteen, ylläpidettävyyteen, suorituskykyyn ja siirrettävyyteen, vaatii vielä täsmällisempää teoreettista jäsentämistä. Jatkotutkimuksessa voitaisiin tarkastella, millä tavalla kestävyys eroaa muista ei-toiminnallisista vaatimuksista ja milloin se toimii niitä kokoavana yläkäsitteenä. Teoreettisen jatkotutkimuksen tavoitteena olisi muodostaa yhtenäisempi käsitteellinen perusta kestävälle ohjelmistokehitykselle.

Lähteet

Abdullateef Shola, O. (2016). Master thesis: Early investigation towards defining and measuring sustainability as a quality attribute in software systems. Lappeenranta University of Technology (LUT).

Ahi, P., & Searcy, C. (2015). Assessing sustainability in the supply chain: A triple bottom line approach. *Applied Mathematical Modelling, 39*(10–11), 2882–2896. https://doi.org/10.1016/j.apm.2014.10.055

Albertao, F., Xiao, J., Tian, C., Lu, Y., Zhang, K. Q., & Liu, C. (2010). Measuring the sustainability performance of software projects. In *2010 IEEE 7th International Conference on E-Business Engineering* (pp. 369–373). IEEE.

Alsaqaf, W., Daneva, M., & Wieringa, R. (2017). Quality requirements in large-scale distributed agile projects: A systematic literature review. In *Requirements Engineering: Foundation for Software Quality* (pp. 219–234). Springer.

Becker, C., Betz, S., Chitchyan, R., Duboc, L., Easterbrook, S. M., Penzenstadler, B., Seyff, N., & Venters, C. C. (2016). Requirements: The key to sustainability. *IEEE Software, 33*(1), 56–65.

Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S., Penzenstadler, B., Seyff, N., & Venters, C. (2015). Sustainability design and software: The Karlskrona Manifesto. In *Proceedings of the 37th International Conference on Software Engineering* (pp. 467–476). IEEE. https://doi.org/10.1109/ICSE.2015.179

Berander, P., Damm, L., Eriksson, J., Gorschek, T., Henningsson, K., Jönsson, P., Kågström, S., Milicic, D., Mårtensson, F., Rönkkö, K., Tomaszewski, P., Lundberg, L., Mattsson, M., & Wohlin, C. (2005). *Software quality attributes and trade-offs*. Springer. https://doi.org/10.1007/s11219-005-4249-7

Berryman, D. R. (2019) Ontology, Epistemology, Methodology, and Methods: Information for Librarian Researchers. Medical Reference Services Quarterly. Vol. 38 (3), 271-279.

Boehm, B., Rosenberg, D., & Siegel, N. (2019). Critical quality factors for rapid, scalable, agile development. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (pp. 514–515). IEEE.

Bourimi, M., Barth, T., Haake, J. M., Ueberschär, B., & Kesdogan, D. (2010). AFFINE for enforcing earlier consideration of NFRs and human factors when building socio-

technical systems following agile methodologies. In *Human-Centred Software Engineering* (pp. 182–189). Springer.

Bourque, P., & Fairley, R. E. (Eds.). (2014). Guide to the Software Engineering Body of Knowledge (SWEBOK Guide): Version 3.0. IEEE Computer Society.

Capra, E., Francalanci, C., & Slaughter, S. A. (2012). Is software "green"? Application development environments and energy efficiency in software products. *Information and Software Technology, 54*(1), 60–71. https://doi.org/10.1016/j.infsof.2011.07.005

Celkee Oy, Tietotekniikan liitto ry, & Ohjelmistoyrittäjät ry. (2013). *Tietojärjestelmien hankinta Suomessa 2013: Tutkimusraportti*. https://tivia.fi/web/content/2029?unique=2e95e7acc8640b793632733b95a62600caeb9006

Charette, R. N. (2018). *The biggest IT failures of 2018*. IEEE Spectrum. [URL puuttuu.]

Dick, M., & Naumann, S. (2010). Enhancing software engineering processes towards sustainable software product design. In *24th International Conference on Informatics for Environmental Protection (EnviroInfo 2010)* (pp. 706–715).

Dreesen, T., Diegmann, P., Binzer, B., & Rosenkranz, C. (2019). Journey towards agility: A retro- and prospective review. In *Proceedings of the 52nd Hawaii International Conference on System Sciences* (pp. 6950–6959).

Du, S., Swaen, V., Lindgreen, A., & Sen, S. (2013). The roles of leadership styles in corporate social responsibility. *Journal of Business Ethics, 114*(1), 155–169. https://doi.org/10.1007/s10551-012-1333-3

Duboc, L., Penzenstadler, B., Porras, J., Koçak, S. A., Betz, S., Chitchyan, R., Leifler, O., Seyff, N., & Venters, C. C. (2020). Requirements engineering for sustainability: An awareness framework for designing software systems for a better tomorrow. *Requirements Engineering, 25*(4), 469–492.

Elkington, J. (1997). *Cannibals with forks: The triple bottom line of 21st century business*. New Society Publishers.

Elkington, J. (2005). Enter the triple bottom line. In A. Henriques & J. Richardson (Eds.), *The triple bottom line: Does it all add up?* Earthscan.

Eskerod, P., & Huemann, M. (2013). Sustainable development and project stakeholder management: What standards say. International Journal of Managing Projects in Business, 6(1), 36–50. https://doi.org/10.1108/17538371311291017

Eskola, J., & Suoranta, J. (1996). *Johdatus laadulliseen tutkimukseen*. Lapin yliopistopaino.

Eriksson, P., & Kovalainen, A. (2016). *Qualitative methods in business research*. SAGE Publications.

European Commission. (2023). *Green digital transformation*. Haettu osoitteesta https://ec.europa.eu

European Parliament and Council of the European Union. (2014). *Directive 2014/95/EU of the European Parliament and of the Council*. EUR-Lex. Haettu 16.5.2026 osoitteesta https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX%3A32014L0095

European Union. (2022). *Directive (EU) 2022/2464 (Corporate Sustainability Reporting Directive)*. EUR-Lex. Haettu 16.5.2026 osoitteesta https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32022L2464

Farid, W., & Mitropoulos, F. (2012). Novel lightweight engineering artifacts for modeling non-functional requirements in agile processes. In *Proceedings of SoutheastCon* (pp. 1–7). IEEE.

Farrugia, B. (2019) WASP (Write a Scientific Paper): Sampling in qualitative research. Early Human Development, 133, 69-71.

Gareis, R., Huemann, M., & Martinuzzi, A. (2013). *Project management and sustainable development principles*. Project Management Institute.

Green Software Foundation. (2021–). *Software Carbon Intensity (SCI) specification*. Haettu osoitteesta https://greensoftware.foundation

Heikkilä, V. T., Damian, D., Lassenius, C., & Paasivaara, M. (2015). A mapping study on requirements engineering in agile software development. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications* (pp. 199–207). IEEE.

Hennink, M., – Kaiser, B. N., (2022) Sample sizes for saturation in qualitative research: A systematic review of empirical tests. *Social Science & Medicine*, 292, 2-10.

Hilty, L. M., & Aebischer, B. (Eds.). (2015). *ICT innovations for sustainability*. Springer. https://doi.org/10.1007/978-3-319-09228-7

Hirsjärvi, S., Remes, P., & Sajavaara, P. (2007). *Tutki ja kirjoita*. Otavan kirjapaino.

Hristov, I., Appolloni, A., & Chirico, A. (2022). The adoption of the key performance indicators to integrate sustainability in the business strategy: A novel five-dimensional framework. Business Strategy and the Environment, 31(7), 3216–3230. https://doi.org/10.1002/bse.3072

Husgafvel, R., Pajunen, N., Virtanen, K., Paavola, I.-L., Päällysaho, M., Inkinen, V., Heiskanen, K., Dahl, O., & Ekroos, A. (2015). Social sustainability performance indicators: Experiences from process industry. *International Journal of Sustainable Engineering, 8*(1), 14–25. https://doi.org/10.1080/19397038.2014.977373

IFRS Foundation. (2022). *ISSB unanimously confirms Scope 3 GHG emissions disclosure requirements with strong application support, among key decisions*. Haettu 18.2.2025. https://www.ifrs.org/news-and-events/news/2022/10/issb-unanimously-confirms-scope-3-ghg-emissions-disclosure-requirements-with-strong-application-support-among-key-decisions/

Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. *Computers in Human Behavior, 51*, 915–929.

ISO. (2013). *ISO/IEC 14040: Environmental management: Life cycle assessment: Principles and framework*. International Organization for Standardization.

ISO/IEC/IEEE. (2017). ISO/IEC/IEEE 12207:2017: Systems and software engineering — Software life cycle processes. International Organization for Standardization.

ISO. (2018). *ISO/IEC 30170: Information technology: Programming languages: Ruby*. International Organization for Standardization.

Kiger, M. E., & Varpio, L. (2020). Thematic analysis of qualitative data: AMEE Guide No. 131. *Medical Teacher, 42*(8), 846–854.

Kivilä, J., Martinsuo, M., & Vuorinen, L. (2017). Sustainable project management through project control in infrastructure projects. International Journal of Project Management, 35(6), 1167–1183. https://doi.org/10.1016/j.ijproman.2017.02.009

Kivimäki, L., Partanen, L., Porras, J., Tarkkanen, K., Tuikka, A.-M., & Mäkelä, J.-M. (2024). Building Up Green Software Life Cycle Model. 2024 10th International Conference on ICT for Sustainability (ICT4S), 20–28. https://doi.org/10.1109/ICT4S64576.2024.00012

Labuschagne, C., & Brent, A. C. (2005). Sustainable Project Life Cycle Management: The need to integrate life cycles in the manufacturing sector. International Journal of Project Management, 23(2), 159–168. https://doi.org/10.1016/j.ijproman.2004.06.003

Lago, P., Koçak, S. A., Crnkovic, I., & Penzenstadler, B. (2015). Framing sustainability as a property of software quality. *Communications of the ACM, 58*(10), 70–78. https://doi.org/10.1145/2818995

Langer, M. E., & Schön, A. (2003). *Enhancing corporate sustainability: A framework based evaluation tools for sustainable development*. Forschungsschwerpunkt Nachhaltigkeit und Umweltmanagement, Wirtschaftsuniversität Wien.

Lauesen, S. (2020). IT project failures, causes and cures. *IEEE Access, 8*, 72059–72067.

Leffingwell, D. (2010). *Agile software requirements: Lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley.

Liker, J. K. (2004). The Toyota way: 14 management principles from the world's greatest manufacturer. McGraw-Hill.

Lindgren, B.-M. – Lundman, B. – Graneheim, U. H. (2020) Abstraction and interpretation during the qualitative content analysis process. International Journal of Nursing Studies. 108.

Lozano, R. (2008a). Developing collaborative and sustainable organisations. *Journal of Cleaner Production, 16*(4), 499–509. https://doi.org/10.1016/j.jclepro.2007.01.002

Lozano, R. (2008b). Envisioning sustainability three-dimensionally. *Journal of Cleaner Production, 16*(17), 1838–1846.

Lozano, R. (2011). Addressing stakeholders and better contributing to sustainability through game theory. *Journal of Corporate Citizenship, 43*, 45–62.

Lozano, R. (2012a). Are companies planning their organisational changes for corporate sustainability? An analysis of three case studies on resistance to change and their strategies to overcome it. *Corporate Social Responsibility and Environmental Management*. https://doi.org/10.1002/csr.1290

Lozano, R. (2012b). Towards better embedding sustainability into companies' systems: An analysis of voluntary corporate initiatives. *Journal of Cleaner Production, 25*, 14–26. https://doi.org/10.1016/j.jclepro.2011.11.060

Marcelino-Sádaba, S., González-Jaen, L. F., & Pérez-Ezcurdia, A. (2015). Using project management as a way to sustainability. From a comprehensive review to a framework definition. *Journal of Cleaner Production*, 99, 1–16. https://doi.org/10.1016/j.jclepro.2015.03.020

Marimuthu, C., & Chandrasekaran, K. (2017). Software engineering aspects of green and sustainable software: A systematic mapping study. In *Proceedings of the 10th Innovations in Software Engineering Conference* (pp. 34–44). ACM. https://doi.org/10.1145/3021460.3021464

Microsoft. (2014). *Chapter 16: Quality attributes*. Microsoft MSDN. [URL puuttuu.]

Moraga, G., Huysveld, S., Mathieux, F., Blengini, G. A., Alaerts, L., Van Acker, K., De Meester, S., & Dewulf, J. (2019). Circular economy indicators: What do they measure? *Resources, Conservation and Recycling, 146*, 452–461. https://doi.org/10.1016/j.resconrec.2019.03.045

Murugesan, S. (2008). Harnessing Green IT: Principles and practices. *IT Professional, 10*(1), 24–33.*IEEE.* https://doi.org/10.1109/MITP.2008.10

Myhrvold, N. (1997). The future of software, the software industry, and Windows '47. In *ACM97: The Next 50 Years of Computing* (p. 1). Association for Computing Machinery.

Naumann, S., Kern, E., & Dick, M. (2014). Classifying Green Software Engineering – The GREENSOFT Model. Softwaretechnik-Trends, 33(2), 18–19. https://doi.org/10.1007/s40568-013-0027-z

Naumann, S., Dick, M., Kern, E., & Johann, T. (2011). The GREENSOFT model: A reference model for green and sustainable software and its engineering. *Sustainable Computing: Informatics and Systems, 1*(4), 294-304. https://doi.org/10.1016/j.suscom.2011.06.004

Oakland, J. S. (2014). *Total quality management and operational excellence: Text with cases* (4th ed.). Routledge.

OpenSSF, & Cloud Native Computing Foundation. (2024). *Sustainable open-source software guidelines*. Haettu osoitteesta https://openssf.org

Partanen, L., Sipilä, A., Haque, M. S., & Porras, J. (2025). *Mapping of the system of software-related emissions and shared responsibilities* (Version 1) [Preprint]. arXiv. https://doi.org/10.48550/ARXIV.2512.13474

Partanen, L., Sipilä, A., & Porras, J. (2023). Energy consumption and CO2 emissions of a software: Who is responsible? In *ICSOB '23: 14th International Conference on Software Business*. CEUR-WS.org.

Penzenstadler, B. (2013). *What does sustainability mean in and for software engineering? ResearchGate.*

Penzenstadler, B., Raturi, A., Richardson, D., & Tomlinson, B. (2014). Safety, security, now sustainability: The nonfunctional requirement for the 21st century. *IEEE Software, 31*(3), 40–47.

Penzenstadler, B., Raturi, A., Richardson, D., & Calero, C. (2018). Sustainability in software engineering: A systematic literature review. In *2015 IEEE/ACM 4th International*

Workshop on Green and Sustainable Software (GREENS) (pp. 1–10). IEEE. https://doi.org/10.1109/GREENS.2015.13

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley.

Ramesh, B., Cao, L., & Baskerville, R. (2010). Agile requirements engineering practices and challenges: An empirical study. *Information Systems Journal, 20*(5), 449–480. [Huom. tarkista vuosiluku, jos tekstissä on käytetty vuotta 2007 tai 2009.]

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering, 14*(2), 131–164.

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes, 35*(3), 8–13. https://doi.org/10.1145/1764810.1764814

Sabini, L., Muzio, D., & Alderman, N. (2019). 25 years of "sustainable projects": What we know and what the literature says. *International Journal of Project Management, 37*(6), 820–838.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering, 25(4), 557–572. https://doi.org/10.1109/32.799955

Siebenhüner, B., & Arnold, M. (2007). Organizational learning to manage sustainable development. *Business Strategy and the Environment, 16*, 339–353.

Silvius, A. J., & Schipper, R. P. (2014). Sustainability in project management: A literature review and impact analysis. *Social Business, 4*(1), 63–96.

Sipilä, A., Partanen, L., & Porras, J. (2023). Carbon footprint calculations for a software company: Adapting GHG Protocol scopes 1, 2 and 3 to the software industry. In *Software Business: 14th International Conference, ICSOB 2023, Lahti, Finland, November 27–29, 2023, Proceedings*. Springer Nature.

Stacey, R. D. (1993). *Strategic management and organisational dynamics*. Pitman Publishing.

Tate, K. (2005). *Sustainable software development: An agile perspective*. Addison-Wesley Professional.

Teoh, R., Schumann, U., Gryspeerdt, E., Shapiro, M., Molloy, J., Koudis, G., Voigt, C., & Stettler, M. E. J. (2022). Aviation contrail climate effects in the North Atlantic from 2016 to 2021. Atmospheric Chemistry and Physics, 22(16), 10919–10935. https://doi.org/10.5194/acp-22-10919-2022

University of Melbourne. (2011). *Software development lifecycle*. Department of Computer Science. https://www.unimelb.edu.au/accessibility/users/development

Venters, C. C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S., Nakagawa, E. Y., Becker, C., & Carrillo, C. (2018). Software sustainability: Research and practice from a software architecture viewpoint. *Journal of Systems and Software, 138*, 174–188. https://doi.org/10.1016/j.jss.2017.12.026

Vuorinen, L. (2024). Chapter 3: Value creation in projects. In L. Vuorinen & M. Martinsuo (Eds.), *Management of project business* (pp. 28–40). Tampere University.

Vuorinen, L., & Martinsuo, M. (2024). Chapter 1: Introduction to management of project business. In L. Vuorinen & M. Martinsuo (Eds.), *Management of project business* (pp. 5–13). Tampere University. https://urn.fi/URN:ISBN:978-952-03-3242-6

Wagner, S., Méndez Fernández, D., Kalinowski, M., & Felderer, M. (2018). Agile requirements engineering in practice: Status quo and critical problems. *CLEI Electronic Journal, 21*(1), Article 15.

Watson, R. T., Boudreau, M.-C., & Chen, A. J. (2008). *Green IS: Building sustainable business practices*. Global Text Project.

Williams, T., Vo, H., Samset, K., & Edkins, A. (2019). The front-end of projects: A systematic literature review and structuring. *Production Planning & Control, 30*(14), 1137–1169.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). Experimentation in software engineering. Springer. https://doi.org/10.1007/978-3-642-29044-2

World Commission on Environment and Development. (1987). *Report of the World Commission on Environment and Development: Our common future*.