# Vihreä koodi

Toinen, laajennettu laitos

Janne Kalliola, Exove

Esipuhe professori Jari Porras, LUT-yliopisto

EXOVE

# Vihreä koodi

Toinen, laajennettu laitos

Versio 2023-08-31

**Älä tulosta tätä kirjaa.**
Tarvittavan paperin hiilijalanjälki on noin 600 $gCO_2eq$.[1]

Ensimmäinen painos ilmestyi 2022.

Kansikuva: Niilo Isotalo / Unsplash

ISBN 978-952-65306-0-4 (pehmeäkantinen)
ISBN 978-952-65306-1-1 (PDF)

[1] www.sciencedirect.com/science/article/abs/pii/S0959652611004409

# Janne Kalliola

**Janne Kalliola** on Exoven perustajia ja toimii nykyään yrityksen kasvujohtajana.

Hän on lisäksi Koodia Suomesta ry:n hallituksen puheenjohtaja ja käynnistänyt hiilineutraaliusmerkin kehityksen.

Janne on koodannut vuodesta 1983 lähtien, julkaissut nuoruudessaan useita kaupallisia ohjelmistoja niin Suomessa kuin kansainvälisestikin.

Hän on aktiivinen esiintyjä ja on puhunut sekä avoimen lähdekoodin tapahtumissa että vihreän ICT:n tilaisuuksissa.

Vihreä koodaaminen ja yleinen ekotehokkuus on kiehtonut Jannea jo vuosia ja hän puhuu ja konsultoi aktiivisesti asian tiimoilta.

kallio.la

linkedin.com/in/jannekalliola

twitter.com/plastic

# Exove

**Exove** on suunnittelu- ja ohjelmistoyritys, jossa analyyttisyys ja teknologiaosaaminen yhdistyvät inhimilliseen ymmärrykseen. Yritys keskittyy luomaan digitaalisia ratkaisuja, joiden avulla taistellaan digiturhautumista vastaan. Olemme hiilineutraali yritys.

Merkittävimpiä asiakkaitamme ovat Neste, Sanoma, Loiste, Rukakeskus, Oulun yliopisto, Itä-Suomen yliopisto, LUT-yliopisto, Jyväskylän ammattikorkeakoulu sekä Tampereen ja Jyväskylän kaupungit.

exove.com

linkedin.com/company/exove

twitter.com/exove

Exove on osa kotimaista PunaMusta Media Oyj -konsernia, jonka palveluksessa on noin 810 eri alojen osaajaa. Konsernin liikevaihto oli vuonna 2022 133,5 miljoonaa euroa. PunaMusta Media on sitoutunut tieteeseen perustuvien ilmastotavoitteiden (SBT) asettamiseen.


# Exoven vastuullisuuskompassi

Olemme kehittäneet vastuullisuusohjelman, Vastuullisuuskompassin, joka ohjaa sisäisiä ja ulkoisia toimintatapojamme sekä sosiaalisen että ympäristövastuun saralla.

Haluamme edistää vastuullisuuttamme luomalla ja kehittämällä parhaita käytäntöjä alallemme.

Kompassimme neljä suuntaa kattavat vaikutuksemme sisäisesti, ja ulkoisesti asiakkaiden ja kumppanien kanssa.

1. Vastuullinen palvelumuotoilu ja -kehitys
2. Sosiaalinen vastuullisuus
3. Ympäristövastuu
4. Vastuullinen hallinto

[exove.com/fi/vastuullisuus/](exove.com/fi/vastuullisuus/)


# Sisällys

# 1 Esipuhe

*“We must change almost everything in our current societies.*
*The bigger your carbon footprint - the bigger your moral duty.*
*The bigger your platform - the bigger your responsibility.*
*Adults keep saying: 'We owe it to the young people to give them hope.'*
*But I don't want your hope.*
*I don't want you to be hopeful.*
*I want you to panic.*
*I want you to feel the fear I feel every day.*
*And then I want you to act.*
*I want you to act as you would in a crisis.*
*I want you to act as if our house is on fire.*
*Because it is.”*

*— Greta Thunberg, No One Is Too Small to Make a Difference*

Ilmastonmuutos on eittämättä yksi viime vuosien poliittista keskustelua hallinneista aiheista. Viimeistään Pariisin ilmastokokous COP21 vuonna 2015 ja sen päätöslauselma tavoitteesta rajoittaa ilmaston lämpeneminen alle 1,5℃ verrattuna esiteolliseen aikaan käynnisti keskustelun mitä eri toimialat voivat tehdä tämän tavoitteen täyttämiseksi. Samana vuonna julkaistu Global e-sustainability initiativen #SMARTer2030 - ICT Solutions for 21st Century Challenges viitoitti ICT-alan tietä osana tulevaisuuden ratkaisuja. Tämä vihreään koodin pureutuva kirja on luon-

nollinen jatkumo tähän vuodesta 2015 käynnistyneeseen keskusteluun eri toimialojen mahdollisuudesta vaikuttaa ilmastonmuutokseen. Aihe on monestakin syystä erittäin haastava, kuten tulette kirjaa lukiessanne huomaamaan.

Kirjan kirjoittaja Janne Kalliola lähestyy aihetta oman nuoruutensa ohjelmointiympäristöjen kautta. 80-luvulla sekä laitteet että ohjelmointiympäristöt olivat vasta kehitysvaiheessa ja tyypillistä tuolle ajalle olikin resurssien, sekä muisti että prosessoriteho, rajallisuus. Tämä vaikutti suoraan itse koodaamiseen. Vaikka ala ohjelmointikielineen ja -ympäristöineen on huomattavasti kehittynyt noista ajoista, on meillä edelleen ympäristöjä, joissa rajoitteet on huomioitava. Sulautetut ohjelmistot painivat edelleen vastaavien resurssirajoitteiden kanssa ja suurteholaskennassa rajoitteen muodostaa ohjelman suoritusaika. Miksi siis Janne Kalliola tässä kirjassa korostaa vihreän koodin merkitystä ja mitä vihreä koodi oikeastaan on?

Vihreällä koodilla viitataan kirjassa ohjelmistojen energiankulutukseen ja sitä kautta niiden hiilijalan- ja kädenjälkeen. Näkökulma on haastava monestakin syystä.

- Ohjelmiston vaatima energia riippuu monesta eri seikasta kuten kirjassa esitetään. Esimerkiksi algoritmivalinnat, käytetty ohjelmointikieli ja sen ulkoiset kirjastot, valittu ohjelmointiympäristö sekä ajoympäristö kaikki vaikuttavat suoritettavan koodin tehokkuuteen. Tehtävät valinnat ovat itsenäisiä ja vaikuttavat toisiinsa, mikä hankaloittaa optimaalisten valintojen tekemistä.

- Ohjelmiston vaatiman energian mittaaminen on hankalaa niin, että ajoympäristö ei vaikuttaisi tulokseen. Erot silmämääräisesti samanlaisilla laitteilla, esimerkiksi puhelimilla, voivat olla merkittäviä laitteen toteutuksesta riippuen.

- Ohjelmiston energiatehokkuus ei ole vastaavalla tavalla kriittinen tekijä – poislukien ehkä sulautetut järjestelmät – kuin ohjelmistojen alkuaikoina esimerkiksi ohjelmiston koko. Mikäli ohjelmisto käytti liikaa muistia sen ajaminen kohdeympäristössä estyi tai aina-

kin hankaloitui. Kun vielä tavoite on vaikuttaa ohjelmistojen tehokkuuden kautta mahdollisiin päästöihin ja sitä kautta ilmastonmuutokseen, muuttuu vaikutusketju sekä toimintojen että aikajänteen osalta hankalaksi ymmärtää ja havainnoida.

Kirjassa esitetään syitä nykyisen koodin tehottomuudelle ja ratkaisuja sekä suosituksia kuinka koodista voidaan tehdä tehokkaampaa. Näkemykseni mukaan kirjan keskeinen viesti on se, että ohjelmointiympäristön rajoitteiden vähennyttyä ohjelmistojen laatu on aikojen saatossa heikentynyt, mikä näkyy turhana energiakulutuksena. Lainaten kirjan kirjoittajaa "Suomessa koodia tehdään nopeasti, ei nopeaksi". Ohjelmistojen energiatehokkuutta saadaan parannettua heti, kun energiatehokkuudesta tulee ohjelmistoille välttämätön tarve.

Tämä Janne Kalliolan kirjoittama kirja toimii hyvänä keskustelunavauksena aihepiiriin, mutta alan kehityksen kannalta on tärkeää, että keskustelua ja kehitystyötä jatketaan tämän jälkeenkin.

Lappeenrannassa, 23.11.2022

Professori Jari Porras
LUT Yliopisto

# 2 Johdanto

Kun aloitin oman ohjelmointiurani joululahjaksi saamallani Commodore VIC-20:lla, tuotetun koodin piti olla lähtökohtaisesti tehokasta. Laitteen kahdeksanbittisen prosessorin kellotaajuus oli yksi megahertsi ja muistia oli viisi kilotavua. Jos koodi ei ollut tehokasta, niin ei se ollut käytännössä kovin käytettävääkään. Viiteen kilotavuun ei kovin monimutkaisia sovelluksia mahtunut tekemään ja osaamallani Basicilla käytössä oli vain 3,5 kilotavua.

Vertailun vuoksi, puhelimeni 64-bittisen kahdeksanytimisen prosessorin kellotaajuus on 2,9 gigahertsiä ja laitteessa on kahdeksan gigatavua muistia. Prosessoritehoa on siis 23 200 kertaa enemmän – ottamatta edes huomioon bittisyyttä ja modernien prosessorien liukuhihnoja – ja muistia 1,6 miljoonaa kertaa enemmän. Toki saan puhelimellani enemmän aikaan kuin alkuaikojen halvalla tietokoneella.

Jos tilannetta verrataan ensimmäiseen Amigaani, jossa oli jo graafinen käyttöjärjestelmä, kaikki tarvitsemani hyötyohjelmistot ja paljon hienoja pelejä, suhdeluvut eivät juurikaan muutu. Amigassani oli 7,16 megahertsin 16-bittinen prosessori ja 512 kilotavua keskusmuistia. Suhdeluvut kännykkääni ovat nyt 3 318-kertainen prosessoriteho ja 16 384 kertaa muistia.

Moni miettii, että aika on varmasti kullannut muistoni. Se on osittain totta, olivathan ohjelmistot välillä hitaita ja niissä oli vähemmän ominaisuuksia. Tuo mainittu Amiga on edelleen tallessa ja pari vuotta sitten näytin lapsilleni, että millaista tietotekniikka oli siihen aikaan, kun isä lampun osti. Sessio päättyi lapsien poistumiseen omille kännyköilleen ja minä jäin

muutamaksi tunniksi pelaamaan Populousta Amigalla. Pelikokemus oli melko lailla verrannollinen pelaamiini mobiilipeleihin – tosin Populous ei pakottanut minua katsomaan aika ajoin mainosvideoita tai pyytänyt nopeuttamaan edistymistä käyttämällä rahaa.

Yksikään nykyisistä sovelluksista ei toimisi noin vanhalla raudalla. Ne eivät mahtuisi muistiin ja olisivat niin hitaita, että käyttäjän kärsivällisyys loppuisi ennen pitkää. Kuitenkin ennen osattiin tehdä sovelluksia, joilla saatiin suurin piirtein samat asiat aikaan kuin nykyisilläkin. Mistä se tehontarve oikein kumpuaa?

Uusissa ohjelmistoissa on tietysti ominaisuuksia, jotka vaativat nykyisten laitteiden tehoa. Esimerkiksi Adobe Photoshopissa on paljon automaattisia korjaustyökaluja, jotka toimivat hämmentävän hienosti ja säästävät tunteja käsityötä. Samoin tiedostojen koot ovat kasvaneet ja niiden käsittely vaatii lisää muistia ja tehoa.

Moni nykyohjelmisto lienee raskas vain sen takia, että sovellusten tehokkuuteen ei tarvitse kiinnittää erityistä huomiota. Rauta on halpaa ja verkossa riittää siirtokapasiteettia, joten miksi nähdä vaivaa? Koska tämä tilanne on vallinnut jo reilun vuosikymmenen, niin suurta osaa modernista koodista ei ole koskaan optimoitu pieteetillä. Se on aina toiminut riittävän nopeasti. Optimointi on myös suhteellisen kallista puuhaa eikä välttämättä maksa itseään takaisin rahassa mitattuna. Säästöt jäävät hyvin pieniksi.

Tilanne on nyt kuitenkin muuttumassa. Ilmastonmuutos ja Eurooppaa ravisteleva Venäjän hyökkäyssodan aiheuttama energiakriisi pakottavat tarkastelemaan kaikkea energiankulutusta. Ohjelmistojenkin tulee muuttua muun maailman mukana. Tämä tulee olemaan pitkä, monimutkainen ja varmasti myös kivulias prosessi, koska moni meistä on tottunut tekemään ohjelmistoja eri tavalla kuin tulevaisuudessa niitä tullaan tekemään.

Vihreästä IT:stä ja koodista on puhuttu Suomessa jo kohtuullisen paljon viimeisten parin vuoden aikana. Olen ollut itse äänessä ja laatinut yhdessä muun Koodia Suomesta ry:n hallituksen kanssa ohjelmistoalan hiilineutraaliusmerkkiä kriteereineen. Useampi ohjelmistoyritys on alkanut

puhua vihreästä koodaamisesta ja sen edistämiseksi on luotu Tietoyhteiskunnan kehittämiskeskus ry:n Green ICT -hanke ja -ekosysteemi[2].

Kuitenkaan vielä ei olla määritelty, että millaista se vihreä koodi tai ekotehokkaat järjestelmät ovat. Jokainen katsoo asiaa omasta lähtökohdastaan ja keskustelu on osittain vielä toisten ohi puhumista. Tästä syystä kirjoitin tämän lyhyen kirjan ja nyt vajaa vuosi myöhemmin lisäsin siihen luvun paljon energiaa kuluttavista erillisratkaisuista, kuten tekoälyistä ja kryptosopimuksista.

Kirjan tarkoituksena ei ole yksiselitteisesti määrittää vihreää koodia ja tuottaa kaanonia, vaan olen pyrkinyt keskittymään kuvaamaan ajatusmalleja ja erilaisia ratkaisuja. Näiden avulla kaikki asian parissa toimivat voivat jäsentää tilannettaan ja tarpeitaan sekä ennen kaikkea muuttaa toimintaansa ilmastoystävällisemmäksi.

Suomessa on pitkä demokoodaamisen perinne 90-luvulta ja koodariemme koulutustaso on erittäin hyvä. Meillä on käytännöllistä osaamista napakan ja tehokkaan koodaamisen saralta. Otetaan siis vanhat konstit käyttöön, lisätään niihin modernin maailman vaatimat ratkaisut ja ryhdytään tekemään tehokkaampaa koodia; rivi riviltä, sovellus sovellukselta. Ollaan osa ratkaisua, ei osa ongelmaa.

Espoossa 31.8.2023, Róisín Murphyn Overpoweredin soidessa.

Janne Kalliola

[2] tieke.fi/green-ict-ekosysteemi/

# 3 Miksi koodin tulee vihertää?

Digitalisaation ja tätä aiemmin ATK:n vaikutus toimintojen tehostumiselle on kiistatonta ja nykyistä yhteiskuntaa on vaikea kuvitella ilman kaikkialle ulottuvia ohjelmistoja. Käytämme ohjelmistoja jatkuvasti ja ne ovat myös koko ajan taustalla helpottamassa ja ohjaamassa elämäämme. Ohjelmistojen merkitys on suuri ja nopeasti kasvava myös energiankulutuksessa.

Ilmastonmuutoksen torjumiseksi on syytä kääntää kaikki kivet ja myös ICT-teollisuuden täytyy osallistua talkoisiin. Ohjelmistoja on toteutettu viimeiset parikymmentä vuotta ilman merkittävää huolta tehokkuudesta laitteiden nopeuden kasvaessa riittävästi. Lisäksi pilvestä saa lisätehoa muutamalla napsautuksella tai jopa automaattisesti. Skaalautuminen raudalla on ollut halvempaa kuin koodin optimointi.

Kaikki tiedon käsittely, esittäminen ja siirtäminen kuluttavat energiaa. Tällä hetkellä puhdasta energiaa ei ole riittävästi tarjolla ja uusiutuvien energiamuotojen haasteena on tuotannon suuret vaihtelut. Energian säästäminen on siis järkevää.

Vaikka moni datakeskus toimiikin uusiutuvalla tai jopa hiilineutraalilla energialla, tilanne ei muutu kokonaisuutta katsoen lainkaan. Likaisempaa energiaa, jota edelleen tuotetaan, käytetään muualla. Energian kokonaiskulutuksen vähentäminen ja kulutuksen siirtäminen päiviin, joina uusiutuvaa energiaa on paljon tarjolla, ovat suorastaan välttämättömiä toimia planeettamme kannalta.

## 3.1 Käden- ja jalanjälki

Puhuttaessa erilaisten asioiden vaikutuksista ympäristöön, termi **hiilijalanjälki** tulee useasti vastaan. Se tarkoittaa tuotteen tai palvelun tuottamaa ilmastokuormaa, eli kuinka paljon kasvihuonekaasuja sen elinkaaren aikana syntyy.

Hiilijalanjälkeä mitataan hiilidioksidiekvivalenttina, joka on kasvihuonekaasujen yhteismitta. Sen avulla voidaan laskea eri kasvihuonekaasujen, kuten hiilidioksidin tai metaanin, vaikutukset ilmastonmuutoksen voimistumiseen. Eri kaasut toimivat ilmakehässä eri tavoin ja niille on laskettu kertoimet suhteessa hiilidioksidiin, jotta niitä voidaan arvioida yhteismitallisesti.

Jokaisella energiamuodolla on omanlaisensa hiilijalanjälki, joka muodostuu itse energian tuotannosta sekä kaikista toimista, joita tarvitaan energian tuottamiseen tai sen valmisteluun. Uusiutuva energia ei ole itsessään hiilineutraalia, koska esimerkiksi tuulivoiman myllyt pitää rakentaa, asentaa ja huoltaa.

Energiankulutus ja erityisesti sen kasvu on ongelmallista, koska energiaa tuotetaan edelleen epäpuhtaasti ja uusiutuvien energiamuotojen tuotanto ei riitä vastaamaan energiankulutuksen kasvuun. Moni IT-alan toimija käyttää uusiutuvaa energiaa, mikä kannustaa energian tarjoajia rakentamaan lisää uusiutuvan energian tuotantokapasiteettia. Osa toimijoista myös kompensoi energiankulutuksensa aiheuttamat päästöt, mikä vähentää kyseisen toimijan laskennallisia päästöjä. Vielä oleellisempaa olisi kuitenkin vähentää kulutusta, koska kulutuksen kasvu johtaa edelleen fossiilisten polttoaineiden käyttöön.

Toisaalta ohjelmistot vähentävät päästöjä suoraviivaistamalla tai optimoimalla muuta toimintaa. Tätä kutsutaan **hiilikädenjäljeksi**. IT-ala on osana tuottavuusloikkaa vähentänyt turhia välivaiheita prosesseissa tai minimoinut vaikkapa paperin käyttöä ja tulostamista. Kädenjäljellä on iso positiivinen merkitys maailmalle ja sitä ei tule missään nimessä väheksyä. Sillä ei voida kuitenkaan oikeuttaa ohjelmistojen tehottomuutta ja kasvavaa energiankulutusta. Sama kädenjälki voidaan saada aikaiseksi tehok-

kaasti tai tehottomasti, jolloin tietysti maailman kannalta on fiksua valita tehokas toteutus.

## 3.2 Trendit vievät väärään suuntaan

IT-alalla on tällä hetkellä voimassa useita trendejä, jotka valitettavasti vievät asioita ilmaston kannalta väärään suuntaan:

- **Datan määrän kasvu** – dataa tuotetaan, käsitellään ja säilötään enemmän kuin koskaan ja määrät ovat räjähdysmäisessä kasvussa. Data myös kumuloituu, eli vanhaa dataa ei aina tuhota uuden alta, vaan sitä voidaan pitää tallessa vuosia tai jopa vuosikymmeniä.

- **Ohjelmistotuotannon mittaus** – oman kokemukseni mukaan ohjelmistotuotannon tehokkuutta mitataan, muutamia poikkeuksia lukuunottamatta, enimmäkseen tuotettujen ominaisuuksien määrässä per käytetty aikayksikkö. Ohjelmistojen tehokkuus on epäoleellista, ellei tehokkuuden puute häiritse käyttäjiä liiaksi.

- **Uusien laitteiden himo** – IT-alalla on jatkuva tarve tuottaa uusia versioita laitteista, joissa on aina enemmän tehoa kuin edellisessä sukupolvessa. Laitteiden kapasiteetti, esimerkiksi näytön tarkkuus tai prosessorin nopeus, on täysin riittävä palvelemaan edelleen kivikautista ihmiskehoa, mutta silti laitteita arvioidaan pitkälti tällaisten ominaisuuksien kehittymisen kautta. Tätä ongelmaa pahentaa laitteiden ohjelmistotuen loppuminen tai sovellusten tehovaatimusten kasvu, jolloin vanha laite muuttuu asteittain käyttökelvottomaksi.

- **Siirtyminen mobiiliverkkoihin** – yhä suurempi osa tiedosta siirretään langattomasti ja yhä enemmän asioista hoidetaan kännykällä. Se on kovin kätevää, mutta merkittävästi energiatehottomampaa kuin langallinen yhteys verkkoon. Toisaalta, kännykät ovat tiedon prosessoinnissa energiatehokkaampia kuin tietokoneet, joten asioiden hoito kännykällä voi olla järkevää, jos sillä vältetään tietokoneen käyttöä.

- **Mainosrahoitteisuus** – erityisesti mobiilissa sovellukset ovat pitkälti ilmaisia ja niiden kehitys rahoitetaan mainoksilla, sovelluksen sisäisillä ostoksilla tai näiden yhdistelmällä. Aalto-yliopiston tutkimuksessa[3] on arvioitu mainosverkostojen vievän noin 10 % koko Internetin sähkönkulutuksesta niiden tehdessä jatkuvasti automatisoituja huutokauppoja mainostajien välillä jokaisesta mainospaikasta.

- **Tekoälyn käytön kasvu** – viimeisen vuoden aikana tekoäly on noussut abstraktista käsitteestä jokapäiväiseksi työkaluksi. Päivittäin tulee uusia tekoälyyn pohjautuvia tai koneoppimista hyödyntäviä ratkaisuja, joiden taustalla on usein valtavat datamäärät ja jotka käyttävät merkittävän määrän laskentakapasiteettia. Emme vielä tiedä, tulevatko tekoälyn käytön hyödyt – prosessien ja toimintatapojen tehostuminen – kattamaan tekoälyn käytöstä syntyneen kulutuksen. Vai aukeaako siitä yksi uusi turha tapa kuluttaa suuret määrät energiaa.

Käynnissä on jatkuva ohjelmistojen ja datan määrän turpoaminen, joka vaatii nopeampia laitteita ja verkkoja. Näiden avulla voidaan taas tehdä tehottomampaa koodia tai lisätä datan määrää, joka johtaa taas laitteiden ja verkkojen kehittymiseen. Tämä sykli tulisi uskaltaa katkaista.

Esimerkiksi verkkosivustot ovat kasvaneet vuosi vuodelta. Aalto-yliopisto teki vuonna 2022 selvityksen[4] suosituista suomalaisista verkkosivuista. Mukana oli sekä yrityksiä että julkishallinnon toimijoita, yhteensä noin tuhat erilaista sivustoa. Selvityksessä havaittiin, että sivustoja tuotetaan vaihtelevalla osaamisella. Osa sivustoista on hyvin optimoituja ja siten niiden datamäärä on pieni ja toiset kuluttavat valtavia määriä dataa.

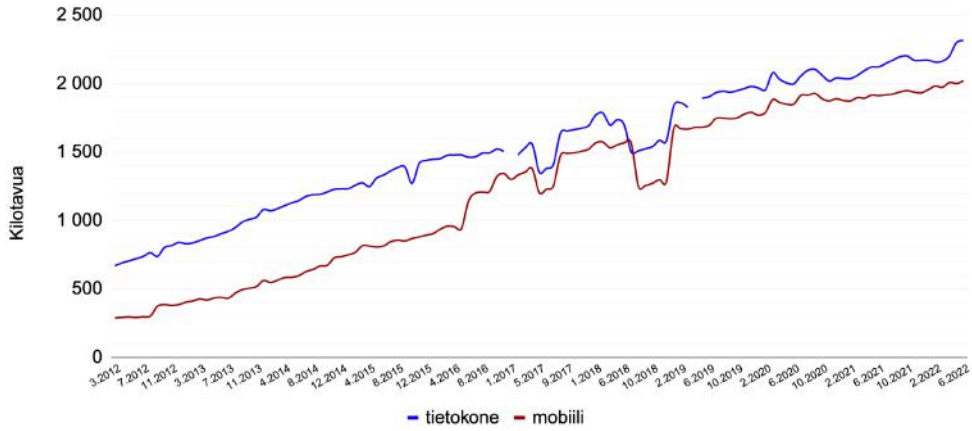
HTTP Archive[5] on tilastoinut sivustojen muutoksia vuosittaiseen Web Almanac -julkaisuunsa. Tilastoista selviää, että viimeisen kymmenen vuo-

[3] www.sciencedirect.com/science/article/pii/S0195925517303505

[4] aaltodoc.aalto.fi/bitstream/handle/123456789/114010/isbn9789526407395.pdf

[5] httparchive.org/

den aikana verkkosivustot ovat kasvaneet kooltaan kolminkertaisiksi[6]. Aikoinaan mobiilisivustot olivat kevyempiä, mutta nykyään ne eivät eroa enää tietokoneille suunnatuista sivustoista, sillä vain niiden ulkoasu muuttuu päätelaitteen mukaan.




*Kuva 1. Sivustojen koon kasvu ajan kuluessa. Lähde: Web Almanac.*

Myös kuluttajien vaatimukset sivustoja ja muita web-palveluja kohtaan ovat kasvaneet. Näyttöjen resoluutioiden kasvaessa ja siirtoverkkojen nopeutuessa kuluttajat ovat tottuneet vaatimaan sisältöjä, jotka näyttävät hyviltä uusilla laitteilla.

Ylempänä mainitun Aalto-yliopiston selvityksen mukaan yli 40 % webbisivujen datasta oli kaikissa mittauksissa kolmansien osapuolien tuottamia resursseja. Tuhannen sivun otoksessa datan määrä jakaantui seuraavasti.

| Tiedostotyyppi | Osuus | Kuvaus |
|---|---|---|
| Kuvat | 56 % | Sivustolla käytettävät kuvat, esimerkiksi valokuvat, taustakuvat, ikonit |
| JavaScript | 22 % | Ohjelmakoodia sivuston |

[6] almanac.httparchive.org/en/2021/page-weight

| Tiedostotyyppi | Osuus | Kuvaus |
|---|---|---|
| | | toiminnallisuuksien, integraatioiden ja käyttäjän seurannan toteuttamiseksi. |
| Mediatiedostot | 6,3 % | Audio- ja videotiedostot sivustolla. |
| Kirjasimet | 4,7 % | Sivustolla käytettävät kirjasimet. Osa ikoneista voi olla myös ladattuna kirjasimena. |
| XHR | 4,5 % | Sivustolla toimivan koodin palvelimelta hakemat tiedostot – voivat olla kuvia, videoita, tekstiä tai erilaisille sovelluksille suunnattuja dokumentteja. |
| CSS | 2,7 % | Sivujen visuaalisen tyylin rakennusohjeet. |
| HTML | 2,0 % | Sivuston rakenteen kuvaus ja tekstisisältö. |
| Muut tiedostot | 1,4 % | Tiedostot, joita ei voitu kategorisoida yllä mainittuihin pääkategorioihin. |

Taulukon luvut on otettu vieritetty tietokone -kategoriasta, eli sivustoa on selattu tietokoneella toimivalla selaimella ja sivu on vieritetty alalaitaan saakka. Osa resursseista ladataan vasta, kun ne ilmestyvät näkyviin (lazy loading), mistä syystä vierittäminen muuttaa eri tiedostojen datamäärien suhteita.

Kannattaa huomata, että sivun tekstisisällön ja rakenteen osuus on vain 1/50 osa sivun datamäärästä. Tästäkin merkittävä osa kuluu rakenteen kuvaamiseen ja muihin tiedostoihin viittaamiseen. Moderneilla sivustoilla uutta tekstisisältöä voidaan ladata myös käyttäjän vierittäessä sivustoa, jolloin kyseiset ladatut sisällöt lasketaan mukaan XHR-tyypin alle, vaikka ne olisivatkin HTML:ää.

Kuvilla ja medialla on tärkeä rooli sivustolla, mutta lähtökohtaisesti suurin osa webbisivuista on edelleen tekstivetoisia ja kävijä käy niillä tekstin vuoksi. Hyötysuhde on huono.

Yli viidennes sivuston datamäärästä tulee erilaisista skriptitiedostoista, joilla sivustosta tehdään miellyttävämpi käyttää ja joilla toisaalta raportoidaan käyttäjän tekemisistä analytiikalle ja mainosjärjestelmille.

## 3.3 Tehokkaat laitteet aiheuttavat harhaa

Sovelluskehittäjien työlaitteet ovat yleensä vain muutaman vuoden vanhoja, erittäin tehokkaita laitteita; keskimäärin yrityskäytössä olevat laitteet uusitaan kolmen vuoden välein leasing-kauden vaihtuessa. Tämä johtaa kehittäjiä harhaan saatavilla olevan tehon suhteen, jolloin sovelluksen optimointia ei tehdä tarpeeksi pitkälle ja vanhemmilla laitteilla palveluja käyttävät ihmiset kärsivät sovellusten hitaudesta, tai sovellukset eivät toimi enää lainkaan laitteella.

Lisäksi taaksepäin yhteensopivien ohjelmistojen teko on kalliimpaa kuin jättää vanhemmat laitteet tukematta. Kaikki syyt johtavat tarpeeseen vaihtaa laite uudempaan ja synnyttää lisää päästöjä laitteiden valmistuksessa ja logistiikassa.

Täytyy samalla kuitenkin todeta, että laitteet ja verkot ovat kehittyneet myös energiapihimmiksi. Tehonlisäys ei välttämättä vaadi vastaavaa sähkönkulutuksen lisäämistä tekniikan edistymisen vuoksi. Toisaalta laitteet voisivat olla merkittävästi energiapihimpiä nykyteknologialla, jos niiltä vaaditut tehotarpeet olisivat myös pienempiä.

Sitran selvitysten mukaan elektroniikkaromu on maailman nopeimmin kasvava romulaji 7 %:n vuosittaisella kasvulla ja siitä vain 17 %:a käsitellään asianmukaisesti vuosittain[7]. Lisäksi on huomioitava, että kierrättäminen itsessään on hyvin energiaintensiivistä toimintaa, koska siinä esimerkiksi sulatetaan metalleja. Vanhaksi tai turhaksi jäänyt laite on parempi kier-

[7] www.sitra.fi/artikkelit/viisi-tarkeaa-kysymysta-digitalisaation-ymparistovaikutuksista/

rättää kuin olla kierrättämättä, mutta vielä parempi vaihtoehto on pyrkiä pidentämään laitteen käyttöikää.

Samaisessa Sitran artikkelissa on ongelman suuruuden konkretisoimiseksi laskettu, että vuonna 2019 tuotetulla elektroniikkajätteellä olisi rakentanut 5 000 Eiffel-tornia.

## 3.4 Euroopan unionin teemat

Euroopan unioni on eräs maailman edistyneimmistä yhteisöistä vastuullisuusasioissa ja 25.11.2021 julkaistussa Euroopan unionin virallisessa lehdessä[8] on kuvattu unionin suosituksia energiatehokkaampien järjestelmien rakentamiseksi. Dokumentin *tavoitteena on siis auttaa kaikkia televiestintä- ja TVT-palvelujen alan organisaatioita keskittymään olennaisiin välittömiin ja välillisiin ympäristönäkökohtiin sekä saamaan tietoa ympäristöasioiden hallinnan parhaista toimintatavoista, asianmukaisista alakohtaisista ympäristönsuojelun tason indikaattoreista, joilla ne voivat mitata ympäristönsuojelun tasoaan, sekä huipputason osaamista koskevista vertailuesimerkeistä* – suora lainaus – ja se antaakin hyvän kuvan asioiden kehityssuunnista EU-alueella.

Tämän kirjasen kannalta kiinnostavat asiat alkavat kohdasta *3.1.6 Tietoliikennetarpeen minimointi vihreiden ohjelmistojen avulla*. Kuvattu paras toimintatapa sisältää seuraavia toimia:

- Kehitetään energiatehokkaampia ohjelmistoja minimoimaan laitteiden virrankulutusta.
- Suunnitellaan ohjelmistot loppukäyttäjätarpeiden mukaan ja kysyntään mukautuviksi, tavoitteena ehkäistä energian liikakulutusta ja laitteiden vanhenemista.
- Seurataan ohjelmistojen energiankulutusta.

[8] eur-lex.europa.eu/legal-content/FI/TXT/PDF/?uri=CELEX:32021D2054&from=EN

- Arvioidaan ohjelmistojen ympäristövaikutuksia elinkaari-arvioinnilla kehitysvaiheessa ja mittaamalla suorituskykyä käyttövaiheessa.
- Refaktoroidaan nykyiset ohjelmistot energiatehokkaammiksi.

Samoin kohta *3.2.2 Tiedonhallintaa ja tallentamista koskevien toimintaperiaatteiden määrittely ja täytäntöönpano* on mielenkiintoinen. Siinä on kuvattu vastaavasti paras toimintatapa:

- Minimoidaan tallennettavan tiedon määrä.
- Maksimoidaan jaettujen alustojen käyttö.
- Yhdistetään olemassa olevia palveluja ja poistetaan tarpeettomia laitteita käytöstä.

Dokumentissa käsitellään lisäksi laajasti datakeskusten ja tiedonsiirtoverkkojen energiatehokkuuden parantamista sekä käyttäjien päätelaitteiden energiankäytön optimointia.

Jokaisessa luvussa kuvataan parhaan toimintatavan lisäksi joukko mittareita (indikaattorit) ja huipputason (“benchmarks of excellence”) vertailuesimerkkejä, joilla oman yrityksen tilannetta voi verrata pitkälle edistyneisiin organisaatioihin. Alla muutama poiminta mittareista:

- Siirretyn datan määrä suhteessa ohjelmiston käyttöön (bittiä verkkosivunäkymää kohti tai bittiä mobiilisovelluksen käyttöminuuttia kohti)
- Niiden uusien ohjelmistojen osuus, joiden osalta energiatehokkuutta on käytetty hankinnan valintaperusteena (%)
- Niiden hiljattain kehitystoimien kohteena olleiden ohjelmistojen osuus, joiden osalta energiatehokkuutta on käytetty ohjelmiston kehittämisperusteena (%)
- Niiden ohjelmistojen osuus, jotka on refaktoroitu tai joiden koodit on tarkistettu energiatehokkuuden parantamiseksi (%)

- Niiden ohjelmistokehittäjien (työntekijöiden) osuus, jotka ovat saaneet koulutusta energiatehokkaista ohjelmistoista (%)

Huipputasolle ei ole vielä vaikea päästä, koska esimerkiksi ohjelmistokehitykseen liittyvistä mittareista on kuvattu seuraavat huipputason tulokset:

- Kaikki työntekijät (ohjelmistokehittäjät) ovat saaneet koulutusta energiatehokkaista ohjelmistoista
- Vuoden aikana on toteutettu ainakin yksi hanke tietoliikennetarpeen minimoimiseksi vihreiden ohjelmistojen avulla

### 3.4.1 Uusi raportointidirektiivi CSRD

Euroopan unioni uudistaa myös omaa lainsäädäntöään yritysten yhteiskuntavastuun raportoinnin suhteen.[9] Uusi raportointidirektiivi CSRD, Corporate Sustainability Reporting Directive, on työn alla ja se tulee korvaamaan nykyisen Non-Financial Reporting -direktiivin (NFRD)[10].

Tavoitteena on korjata voimassa olevien sääntöjen puutteet ja yhdenmukaistaa raportointia vertailuvuuden mahdollistamiseksi. Direktiivin laadinnan taustalla on tarve arvioida yrityksen ympäristö- ja yhteiskuntavaikutuksia ja siten mitata yritystoiminnan kestävyyttä. Liiketoiminnan kestävyydestä raportoidaan erillisellä kestävyysraportilla, joka pohjautuu direktiiviin ja eurooppalaisiin kestävyysraportoinnin standardeihin. Esimerkiksi yrityksen kasvihuonekaasupäästöt ja niiden kehittyminen raportoidaan vuosittain.

Ehdotuksen mukaan suurien ja pörssilistattujen yritysten tulee raportoida vastuullisuusteemoista koneluettavasti osana toimintakertomustaan. Euroopan komissio määrittelee erikseen myöhemmin riskialat, joissa

[9] www.consilium.europa.eu/fi/press/press-releases/2022/02/24/council-adopts-position-on-the-corporate-sustainability-reporting-directive-csrd/

[10] www.iconics.fi/post/kestävyysraportointidirektiivi

raportointivelvollisuus ulottuu myös PK-yrityksiin. Tilintarkastajan tulee tarkastaa raportoitujen tietojen oikeellisuus.

Raportointistandardeja on tulossa näillä näkymin 13 kappaletta ja ne ovat yksityiskohtaisia. Standardeihin ja prosessiin voi tutustua Euroopan tilinpäätösraportoinnin neuvoa-antavan ryhmän, EFRAG:n, sivuilla.[11]

## 3.5 IFRS ja päästöraportointi

Lokakuussa 2022 The International Sustainability Standards Board (ISSB) päätti yksimielisesti vaatia jatkossa IFRS:ää noudattavien yritysten tiedottavan hiilijalanjäljestään sisältäen kaikkien kolmen hiiliraportoinnin laajuusalan (scope) alaiset päästöt[12].

International Financial Reporting Standards (IFRS) on kansainvälinen standardi tilinpäätöstietojen julkaisuun. Se on ollut käytössä kaikissa EU-maissa vuodesta 2005 ja sitä vaaditaan maailmalla kaikkiaan 167 alueella, mukaan luettuna käytännössä kaikki kehittyneet maat[13].

Jatkossa IFRS:n mukaisesti raportoivien yritysten tulee julkistaa omat päästönsä osana tilinpäätöstä. Koska kaikki kolme laajuusalaa ovat mukana, myös IT-järjestelmien energiankulutus ja päästöt tulevat raportoitavaksi.

Vaatimus ei ole voimassa per heti, aikataulu ei ole tällä hetkellä vielä tiedossa. Lisäksi muutokseen tarjotaan varmasti muutamien vuosien siirtymäaika. Erityisesti isojen yritysten on syytä aloittaa päästöjen seuraaminen jo nyt ja ottaa IT-järjestelmien energiankulutus ja päästöt pala palalta haltuun.

---

11 www.efrag.org/lab3

12 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/

13 en.wikipedia.org/wiki/International_Financial_Reporting_Standards

## 3.6 Kehittäjien vastuu

Sovellusten toteutuksen tehokkuus on ensisijaisesti ohjelmistoarkkitehtien ja -kehittäjien vastuulla. Sovellukseen kohdistuvat vaatimukset, jotka kumpuavat yleensä liiketoimintatarpeista, määrittävät pitkälti sovelluksen tehokkuuden tai tehottomuuden – valitettavasti tyypillisesti tehottomuuden.

Mutta vain alan ammattilaiset osaavat toteuttaa sovelluksia ja siksi heillä on avaimet toteuttaa niistä tehokkaita tai tehottomia. Valitettavasti alalla on edelleen harhaluuloja, että asialla ei olisi mitään väliä – esimerkiksi hiilikädenjäljen koon takia – tai asialle ei kannata tehdä mitään, koska joku muu asia on merkityksellisempi. Lisäksi ohjelmistoja ei tällä hetkellä vielä hankita energiatehokkuus mukana joko vaatimuksissa tai valintakriteereissä, mutta tähän ollaan nyt herätty. Luultavasti lähivuosina nähdään nykyistä enemmän toiveita ja vaatimuksia ratkaisun energiatehokkuudesta. Toivon tämän kirjasen osaltaan johtavan hankintakriteerien muuttumiseen.

Olen käynyt keskusteluja, joskus tulisiakin, siitä, että onko sovelluskehittäjän järkevämpää käyttää aikaa sovelluksen optimointiin vai tulla työpaikalle pyörällä oman auton sijasta. Nämä eivät kuitenkaan sulje toisiaan pois, eli myös pyöräilevä koodari pystyy optimoimaan sovellustaan ja fiksut valitsevat molemmat.

Kuten Hämähäkkimiehen setä Ben Parker on eri inkarnaatioissaan todennut: “suuri voima tuo mukanaan suuren vastuun.” [14]

[14] en.wikipedia.org/wiki/With_great_power_comes_great_responsibility

# Tiivistettynä

1. Digitaalisuus on välttämätöntä tehokkuuden kannalta ja täten ohjelmistojen kasvava energiankulutus vaatii huomiota. ICT-alan on syytä huomioida energiatehokkuuttaan torjuakseen omalta osaltaan ilmastonmuutosta.

2. Kriittisenä haasteena on ICT-alan energiankulutus, joka koostuu ohjelmistojen, laitteiden ja tiedonsiirron käyttämästä energiasta. Uusiutuvan energian saatavuuden vaihtelut edellyttävät toimenpiteitä energian säästämiseksi myös ICT-alalla.

3. Trendit kuten datamäärien kasvu, tehottomat ohjelmistot, jatkuvat laitepäivitykset, langattomat yhteydet ja palveluiden mainospohjainen rahoitus heikentävät energiatehokkuutta sekä kasvattavat ympäristövaikutuksia.

4. Energiatehokas ohjelmistokehitys on ratkaisevassa asemassa laitteiden ja verkkojen suorituskykyvaateiden hillitsemiseksi. Kehittäjillä on täten keskeinen rooli sovellusten optimoinnissa energiankulutuksen hillitsemiseksi.

5. Euroopan unioni keskittyy kestävyyteen CSRD:n kaltaisten säädösten kautta, minkä lisäksi IFRS-standardi tulee pakottamaan hiiliraportointiin. Ohjelmistojen hiilijalanjäljen laskenta tulee näiden säädösten johdostakäytännössä pakolliseksi.

# 4 Mitä tiedämme ja mitä emme

Ohjelmistojen energiankulutus on periaatteessa varsin yksinkertaista. Tiedon käsittely vaatii kellosyklejä prosessorilta ja tiedon siirtäminen käsiteltäväksi esimerkiksi muistista, verkosta tai massamuistista vaatii sekä prosessori- että IO-tehoa.

Hankalaksi asian tekee kuitenkin ohjelmistojen monimutkaisuus, kerroksellisuus ja mittauspisteiden puute. Yhdessä laitteessa on helposti satoja prosesseja käynnissä kuluttamassa energiaa ja jokainen näistä on arkkitehtuuriltaan, algoritmeiltaan ja dataltaan omanlaisensa. Vaikka laitteen käyttämä energia voidaan mitata siihen syötetyn sähkötehon määrästä, niin sen jakautuminen eri osien kesken on epäselvää ja vaikeasti mitattavissa.

Lisäksi monimutkaisuus kertautuu, kun ympäristössä onkin useita laitteita, esimerkiksi käyttäjän päätelaite, edustapalvelin, tietokantapalvelin, erilaisia välimuistipalvelimia ja verkkolaitteita. Kaikissa niissä on omat ohjelmistonsa omine erityispiirteineen. Tähän vielä päälle tulevat erilaiset abstraktiokerrokset, kuten virtuaalikoneet, kontit ja mikropalveluarkkitehtuurin mukaiset toteutukset.

Useasta lähteestä syntyvä mittausdata on tyypillisesti myös yhteismitatonta, jolloin lukujen laskeminen yhteen ei johdakaan luotettaviin lopputuloksiin. Mitattomuus voi syntyä erilaisesta tavasta mitata, tulkita mittauksia tai mittaukset voivat sisältää keskenään erilaisia kohteita – eli toisaalla tietyt asiat sisällytetään mittaukseen ja toisaalla taas ei.

Tästä monimutkaisuudesta johtuen tarkkojen ja luotettavien mittaustulosten hankinta on vaikeaa tai jopa mahdotonta. Osa järjestelmistä voi olla myös kolmansien osapuolten tarjoamia palveluja, joiden energiankulutus on ainoastaan tarjoajan sisäistä tietoa – jos heilläkään edes on sitä.

Perinteisesti prosesseja parannetaan juuri mittausdatan avulla. Vaikka mittauksia ei olisikaan olemassa tai ne olisivat pahasti puutteellisia, ohjelmistojen energiatehokkuutta ja vihreyttä voidaan kehittää. Ja niin tulee tehdäkin, koska meillä ei ole aikaa tai varaa jäädä odottamaan mittausratkaisujen kehittymistä.

Tämä kirja antaa erilaisia ajattelumalleja, joilla ongelmaa voi lähestyä eri kulmilta ja joiden avulla tilannetta voidaan aidosti parantaa. Yksittäinen ratkaisu ei sovi kaikkeen, mutta kun sovelluskehittäjillä on käytössään riittävä työkalupakki miettiä tehokkuutta, muutoksia saadaan kuitenkin aikaiseksi. Oleellista on järjestää riittävästi aikaa tämän työkalupakin käyttämiseen, mikä vaatii energiatehokkuuden priorisointia joidenkin muiden asioiden edelle.

Mittausdata tulee paranemaan tulevina vuosina. Jo nyt esimerkiksi pilvipalveluiden energiankulutuksesta tai hiilijalanjäljestä voi hakea tietoa palveluntarjoajan käyttöliittymästä, mutta myös prosessorivalmistajat tarjoavat mittauslaitteita tehonkäytön mittaamiseen.

Edelleen energiankulutustiedon julkaisemisessa on tietty stigma – sen pelätään olevan yrityksen maineelle haitallista. Mitä useampi taho julkaisee tietojaan, sitä helpommaksi tulee vertailla oman ratkaisun energiatehokkuutta toisiin vastaaviin. Vielä emme kuitenkaan ole tuossa pisteessä, joten tämä kirja keskittyy enemmän muutokseen kuin mitattavuuteen.

Jos sinulla on kuitenkin mahdollisuus mitata energiankulutusta riittävän luotettavasti, se mahdollisuus kannattaa ehdottomasti käyttää ja ottaa osaksi omaa tuotantoprosessia. Näin voidaan esimerkiksi varmistaa, että uusi versio ohjelmistosta ei romahduta tai edes huononna energiatehokkuutta. Mikäli mittausta ei ole mahdollista tehdä, samansuuntaista arviointia voi tehdä esimerkiksi palvelun vasteaikoja seuraamalla: pidentynyt vasteaika viittaa tehontarpeen kasvuun.

Suosittelen lisäksi vahvasti julkaisemaan mahdollisimman paljon relevantteja mittaustietoja, koska ymmärryksemme tehokkuudesta tai sen puutteesta on tiedon puutteen takia hyvin vajavainen.

## Tiivistettynä

1. Ohjelmiston energiankulutus pohjautuu periaatteessa tietojen käsittelyn ja siirron aiheuttamaan kulutukseen. Usein ohjelmistojen monimutkaisuus ja kerroksellisuus tekevät kuitenkin tarkan mittaamisen haastavaksi.

2. Mittaamisen monimutkaisuus moninkertaistuu, kun mukana on useita laitteita ja erilaisia abstraktiotasoja, jotka synnyttävät myös yhteismitallisuushaasteita.

3. Vaikka mittaus onkin haasteellista, ohjelmiston energiatehokkuuden parantaminen on ratkaisevan tärkeää hiilidioksidipäästöjen vähentämiseksi.

4. Julkista mittausdataa on saatavilla hyvin vähän. On siis tärkeää pyrkiä julkaisemaan omat mittaukset alan energiankulutuksen läpinäkyvyyden kasvattamiseksi.

5. Tarkka energiankulutuksen mittaus tulisi integroida tuotantoprosessiin aina sen ollessa mahdollista. Jos tämä ei ole mahdollista, vaihtoehtoiset lähestymistavat – kuten vasteaikojen seuranta – voivat auttaa arvioimaan energiatehokkuutta.

# 5 Modernien ohjelmistojen energiankulutusmalli

Leijonanosa nykyisin kehitettävistä ohjelmistoista pohjautuu asiakas–palvelin -malliin, eli verkossa on palvelin, joka tarjoaa erilaisia palveluja päätelaitteessa toimivaan sovellukseen. Esimerkiksi käytännössä kaikki selaimella käytettävät palvelut ovat tällaisia, samoin suurin osa mobiilisovelluksista ja enenevä määrä työpöytäsovelluksista – joissa tyypillisin käyttötapaus on lisenssien tai tilauksen voimassaolon varmistaminen.

Yksityiskohdat voivat vaihdella merkittävästi erilaisten ohjelmistojen välillä, mutta energiankulutuksen kannalta malli voidaan tiivistää kolmeen osaan:

1. **Palvelinkeskuksessa toimivien ohjelmistojen energiankulutus.** Täällä on varsinainen sovelluspalvelin, jossa on ohjelmiston keskitetty liiketoimintalogiikka ja joka huolehtii yhteydenpidosta päätelaitteessa olevan sovelluksen kanssa. Lisäksi käytössä on tallennus- ja tietokantapalvelimia ja mahdollisesti erilaisia apupalvelimia, kuten varmennus- ja palautusratkaisuja.

   Pilviympäristössä toimiva ohjelmisto voi olla rajoiltaan häilyvämpi nojautuessaan pilviympäristön valmiisiin palveluihin, jotka ovat samalla olennainen osa ohjelmiston ajoympäristöä.

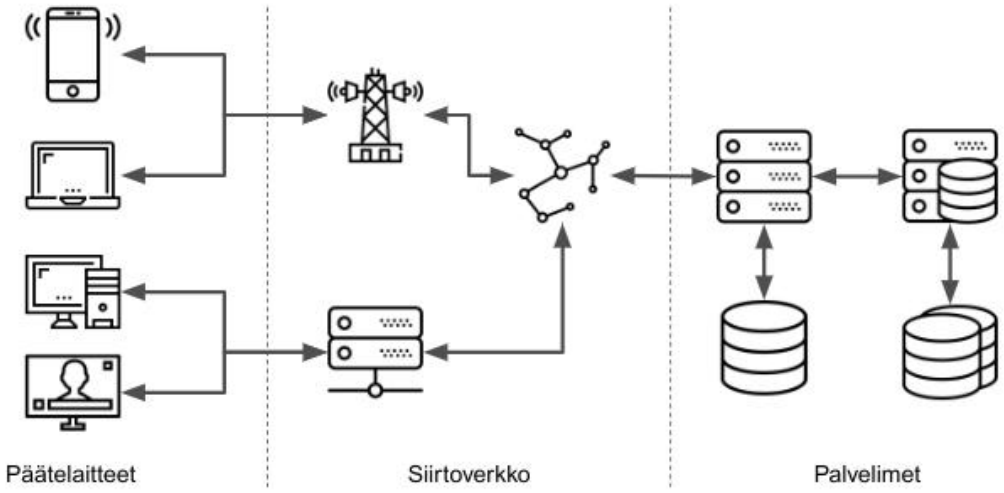
   Palvelinkeskuksen sisäinen tiedonsiirto lasketaan tähän samaan momenttiin.

2. **Tiedonsiirto palvelinkeskuksen ja päätelaitteen välillä.** Tiedonsiirrossa kuluva energia on laskennan kannalta helpointa ajatella lineaariseksi suhteessa datan määrään. Siirrettävän tiedon sisältö voi vaihdella hyvin paljon erilaisten sovellusten välillä, mutta energiankulutus on tavu tavulta sama datan kulkiessa samaa siirtotietä. Siirtoteiden välillä on taas merkittäviä tehoeroja.

   Tiedon käsittely palvelimessa, esimerkiksi siirtokuntoon laittaminen, pakkaus, salaaminen ja purkaminen kuuluvat palvelinkeskuksen kulutukseen ja vastaavasti samat toimet päätelaitteessa kuuluvat päätelaitteen kulutukseen.

3. **Käyttäjän päätelaitteessa käyttämä osa sovelluksesta.** Päätelaitteessa toimiva ohjelmisto voi olla hyvin yksinkertainen ja staattinen webbisivu tai monipuolinen ja -mutkainen päätelaitteeseen asennettu sovellus.

   Energiankulutuksen laskennan kannalta on sinänsä sama, että ajetaanko laitteessa sitä varten kirjoitettua natiivisovellusta vai toimiiko sovellus selaimen sisällä. Jälkimmäisessä tapauksessa selaimen kuluttama energia lasketaan mukaan osaksi sovelluksen energiankulutusta.




*Kuva 2. Modernien ohjelmistojen kolme energiankulutuksen pääaluetta.*

Kaikkien IT-järjestelmien tehtävänä on käsitellä tietoa ja esittää sitä käyttäjälle ymmärrettävässä muodossa. Tiedonkäsittely voi sijaita sekä palvelinkeskuksessa että päätelaitteessa. Käsittelyn sijainti muuttaa siirrettävän tiedon määrää ja mahdollisesti laskentakertojen määrää merkittävästikin.

Sovelluksen arkkitehtuurin suunnittelussa onkin syytä pyrkiä tasapainoiseen ratkaisuun, joka minimoi sekä käsittelyn että siirron ottamalla huomioon jokaisen osa-alueen toteutuksen erityispiirteet energiankulutuksen kannalta. Tämä kuulostaa yksinkertaiselta, mutta käytännön tasolla ongelma ei ole mitenkään triviaali. Lisäksi suunnittelussa voi käydä kuten entisessä itänaapurissa, Viktor Tshernomyrdinin sanoin: “Kyllä me parhaamme yritimme, mutta kävi taas kuten aina.”

Erityisen vaarallista on keskittyä optimoimaan vain yhtä aluetta kahden muun kustannuksella, ellei tähän ole erittäin painavaa syytä, esimerkiksi tiedonsiirron kaista on hyvin kapea tai kustannukset merkittävät – vaikkapa kommunikointi avaruusluotainten kanssa. Helposti nimittäin käy niin, että tehoa tarvitaankin lisää muualla tai käyttökokemus muuttuu huonoksi ja käyttäjät kaikkoavat palvelusta.

Käsitellään tämän kolmikon osat jokainen tarkemmin erikseen ja lisäksi vielä muutamia tällaista yksinkertaista ja selkeää kuvaa monimutkaistavia toteutuksia. On syytä pitää mielessä, että nämä kaikki ovat karkeita yleistyksiä ja jokaisen ohjelmiston erityispiirteet pitää huomioida kyseistä ohjelmistoa optimoitaessa. Näiden mallien avulla voidaan kuitenkin avata keskustelu ja syventyä askel askeleelta omissa ratkaisuissa olevien piirteiden ymmärtämiseen.

## 5.1 Palvelinkeskus ja pilvipalvelut

Palvelinkeskuksessa on tyypillisesti sovelluksen keskitetyt toiminnallisuudet ja tietojen tallennus. Sovellus voi olla yksinkertainen yhdessä fyysisessä laitteessa toimiva palvelu tai useisiin kymmeniin tai satoihin laitteisiin levittäytyvä monipuolinen ratkaisu. Energiankulutuksen kannalta kaikkia käsitellään samalla tavalla, eli sovelluksen käyttämän energian

määrä on sen käyttämien laitteiden ja niiden välisen tiedonsiirron energiankulutuksen summa. Osa laitteista on jaetussa käytössä, mikä hankaloittaa laskentaa.

Tässä kirjassa viitataan “palvelinkeskuksella” sekä fyysisiin palvelinkeskuksiin että pilvessä sijaitseviin palvelimiin, jotka kuitenkin lopulta sijaitsevat yhdessä tai useammassa palvelinkeskuksessa. Pilven erityispiirteet toki nostetaan esille, mikäli ne eroavat merkittävästi perinteisempien palvelinmallien toiminnasta energiatehokkuuden kannalta.

Sovellusta kehittävä taho harvoin omistaa laitteita, vaan ne vuokrataan käyttöön palvelinkeskukselta. Suuret toimijat omistavat omia palvelinsalejaan, mutta periaate on sinänsä sama. Laitteiden tuotanto, logistiikka, huolto ja käytöstä poisto synnyttävät kaikki päästöjä laitteiden kuluttaman energian lisäksi.

Energiansäästön ja päästöjen pienentämisen kannalta onkin oleellista pyrkiä minimoimaan käytettävien laitteiden määrä ja maksimoimaan käytössä olevien laitteiden kuormitus mahdollisimman lähelle ideaalitasoa. Laitteiden ylikuormitus ei kuitenkaan ole järkevää, koska se johtaa pyyntöjen jonoutumiseen, käsittelyaikojen pitenemiseen sekä laitteiden eliniän lyhenemiseen.

Kannattaa huomata, että palvelun hajauttaminen usealle fyysiselle laitteelle lisää myös muiden laitteiden tarpeita. Tarvitaan esimerkiksi kuormanjakajia tai jaettuja levypakkoja, jotka kuluttavat omalta osaltaan energiaa. Hajautus monimutkaistaa sovellusta, koska kaikki tiedon käsittely ei tapahdukaan enää yhteisessä muistiavaruudessa ja sovelluksen pitää varautua tarkemmin tiedon konsistenttiusvaatimuksiin.

Modernit ajoympäristöt ovat virtualisoituja tai kontitettuja. Näissä sovellukselle luodaan oma yksityinen ajoympäristö, joka on täysin sovelluksen käytössä ja mukautettu sovelluksen tarpeisiin. Fyysiseen laitteeseen asennetaan kuitenkin useampi tällainen virtuaalikone tai kontti, jolloin sen kapasiteettia jaetaan usealle sovellukselle.

Nämä välikerrokset syövät oman, hyvin pienen osansa fyysisen laitteen kapasiteetista. Vastavuoroisesti ne mahdollistavat fyysisten laitteiden käy-

tön mahdollisimman tehokkaasti ja voivat tarjota myös automaattista tehojoustoa. Tutkimuksen mukaan konttiteknologia ei merkittävästi vähennä prosessoinnin tai muistin käytön tehoa. Tehoa häviää enemmän tiedonsiirrossa, jossa virtualisointi ja kontitus lisäävät prosessorin käyttöä tiedon kulkiessa useamman ohjelmistokerroksen läpi[15].

Vastaavasti sovellus voidaan rakentaa mikropalveluina, jolloin jokainen toiminto on oma palvelunsa omassa ympäristössään ja nämä palvelut keskustelevat keskenään tekemällä kutsuja toisilleen verkon yli. Ratkaisun etuna on skaalautuminen – jokainen mikropalvelu voidaan periaatteessa ajaa missä tahansa laitteessa palvelinkeskuksessa – ja yksittäisten palveluiden yksinkertaisuus. Haasteena on puolestaan palvelun rakenteen hajaantuminen useaan pienempään osaseen ja energiankulutuksen hämärtyminen. Mikropalvelut helpottavat konesali- ja pilvipalvelujen tarjoajien mahdollisuuksia kohdistaa resursseja optimaalisella tavalla, mikä taas vähentää energiankulutusta kokonaisuudessaan.

## 5.2 Siirtotie

Tiedon siirtyminen palvelinkeskuksen ja päätelaitteen välillä ei ole yleisesti sovelluskehittäjien kontrollissa, ainakaan koko matkaltaan. Palvelinkeskus on tyypillisesti kytketty Internetin runkoverkkoon ja se on käytännössä jo valtavien siirrettyjen tietomäärien takia optimoitu energiatehokkaaksi. Palvelinkeskuksen valinnalla voi tietenkin vaikuttaa käytettävän yhteyden energiankulutukseen, mikäli kulutustieto on tarjolla. Lisäksi content delivery network -ratkaisuilla (CDN) voidaan lyhentää siirtotietä kuluttajalta verkossa olevaan sisältöön.

Palvelun käyttäjällä taas on suuri valta siirtotien suhteen, koska he käyttävät palvelua haluamassaan paikassa ja siellä olevalla verkkoyhteydellä. Siirtotien energiatehokkuutta voidaan analysoida kahden oleellisen parametrin kautta:

[15] ieeexplore.ieee.org/document/7095802

1. **Siirtotien fyysinen toteutus.** Erilaiset tavat siirtää tietoa eroavat tehokkuudeltaan merkittävästi toisistaan. Valokuitu on energiatehokkaimpia tapoja siirtää tietoa ja se onkin käytössä runkoverkoissa sekä joidenkin onnellisten osalta myös viimeisenä linkkinä kotiin tai työpaikalle. Toisessa päässä tehokkuusspektriä on mobiilitiedonsiirto, joka on satoja kertoja tehottomampaa kuin siirtäminen valokuidulla.

   Tutkimuksen[16] mukaan Suomessa vuonna 2020 mobiilitiedonsiirto vaati tehoa noin 220 Wh/Gt[17] ja kiinteistä verkoista erään tutkimuksen otannassa tehottomin 0,25 Wh/Gt ja tehokkain 0,02 Wh/Gt[18]. Tehoerot ovat tuhat- tai jopa kymmentuhatkertaisia.

2. **Tiedon siirretty matka.** Mitä pidemmän matkan tietoa täytyy siirtää palvelinkeskuksen ja päätelaitteen välillä, sen enemmän siirto kuluttaa energiaa riippumatta käytetystä siirtotiestä. Mannertenväliset siirtotiet ovat lähtökohtaisesti erittäin tehokkaita, joten siirto Yhdysvalloista Eurooppaan saattaa olla murto-osa verrattuna viimeisen kilometrin matkaan tukiasemasta kännykkään, mutta kuluttaa se silti energiaa.

Lisäksi ilmaston suhteen tulee ottaa huomioon siirtotien käyttämän energian hiilidioksidipäästöt. Ne vaihtelevat sähkön tuotantotavan ja mahdollisten kompensointien mukaan, mutta tyypillisesti tätä tietoa ei ole mistään julkisesti saatavissa. Hyvä arvio saadaan maakohtaisista keskiarvoarvioista, esimerkiksi tämän tekstin kirjoitushetkeä edeltävän kuuden kuukauden aikavälillä, helmikuu–heinäkuu 2023, Suomessa kulutettu säh-

[16] joonasnuutinen.fi/wp-content/uploads/2022/01/Nuutinen2021_A-Comparison-of-the-Energy-Consumption-of-Broadband-Data-Transfer-Technologies.pdf

[17] ficom.fi/ajankohtaista/uutiset/digiratkaisuilla-energiatehokkuuteen-mutta-ei-ilman-sahkoa/

[18] www.prysmiangroup.com/staticres/energy-consumption-whitepaper/26/index.html

kö tuottaa Fingridin mukaan keskimäärin 34 gCO2/kWh päästöjä[19]. Vastaavasti Saksan vuoden 2022 keskiarvo on 385 gCO2/kWh eli päästöt ovat Suomeen verrattuna yli yksitoistakertaiset [20].

Globaalien palvelujen siirtotiet kulkevat eri maiden halki ja aukotonta laskentaa on mahdotonta tehdä. Lisäksi päätelaite on moderneissa sovelluksissa ja erityisesti web-palveluissa tyypillisesti kytkeytynyt useampiin taustajärjestelmiin – esimerkiksi analytiikka, videot tai chat – joista jokainen mahdollisesti käyttää eri siirtoreittejä.

Samoin palvelinkeskuksessa sijaitseva palvelinsovellus voi olla yhteyksissä muihin taustajärjestelmiin siirtoverkon yli. Toki näissä tilanteissa voidaan olettaa, että kyseinen yhteys kulkee vain runkoverkossa.

Siirtotien laatua loppukäyttäjälle asti on siis mahdotonta päätellä aukottomasti palvelinkeskuksesta käsin. Periaatteessa päätelaitteen tyypistä voidaan päätellä osviittaa siirtotiestä, mutta toisaalta pöytäkonekin voi olla verkossa mobiililaajakaistan kautta tai vastaavasti kännykkä kytketty langattomaan lähiverkkoon ja siitä edelleen valokuituverkkoon. Mobiilisovellus voi selvittää kännykän käyttöjärjestelmältä ja sivusto web-selaimelta käytetyn yhteyden laadun, jonka mukaan järjestelmän toimintaa voidaan mukauttaa.

Yhteistä kaikille siirtoteille on kuitenkin siirretyn tiedon määrän vaikutus energiankulutukseen, eli mitä enemmän sovellus siirtää tietoa palvelinkeskuksen ja päätelaitteen välillä, sitä enemmän kuluu energiaa.

Lisäksi tiedonsiirtotarpeiden kasvaessa kasvaa myös kapasiteetti, mistä aiheutuu ilmastopäästöjä. Siirrettävän tiedon minimointi siis parantaa tilannetta sekä vähentää siirtotien käyttämää energiaa ja hidastaa tarvetta nostaa verkon siirtotehoa.

[19] www.fingrid.fi/sahkomarkkinainformaatio/co2/

[20] www.statista.com/statistics/1290224/carbon-intensity-power-sector-germany/

## 5.3 Päätelaite

Osa sovelluksesta on ajossa päätelaitteessa, joko itsenäisenä sovelluksena tai toisen sovelluksen sisällä. Ensimmäisestä tyypillisiä esimerkkejä ovat tietokoneisiin ja kännyköihin asennettavat sovellukset ja toista mallia edustaa esimerkiksi webbisivut ja -palvelut. Näiden välille ei voida vetää selkeää rajaa, koska esimerkiksi asennetut sovellukset saattavat näyttää webbisisältöä osana omaa käyttöliittymäänsä. Vaikka tarkkaa rajaa on vaikea vetää, oleellista on huolehtia kaikkien käytettyjen osien optimoinnista ja sopimisesta tarkoitukseensa.

Sovelluksen tiedon prosessointia voidaan tehdä sekä palvelinkeskuksessa että päätelaitteessa. Prosessoinnin sijainti vaikuttaa, joskus merkittävästikin, siirrettävän tiedon määrään. Valitulla arkkitehtuurilla tulisi pyrkiä minimoimaan kaikkien komponenttien yhteenlaskettu energiankulutus. Mitään yleispätevää mallia tähän ei ole, sillä sovellusten toimintamallit poikkeavat toisistaan suuresti.

Lisäksi päätelaitteiden energiankulutuksissa on merkittäviä eroja. Esimerkiksi kännykät on optimoitu energiapiheiksi, koska niiden koko ja paino rajaavat isommat ja tehokkaammat akut mahdollisuuksien rajojen ulkopuolelle. Käyttäjät kuitenkin haluavat käyttää laitetta koko päivän yhdellä latauksella.

Vastaavaa optimointia ei tarvitse tehdä vaikkapa 65” taulutelevisiolla, joka on kytketty kiinteään sähköverkkoon. Toki EU:n ja muiden vastaavien tahojen energialuokitukset pyrkivät ohjaamaan laitteet mahdollisimman energiatehokkaiksi, mutta erot ovat silti merkittävät. Energialuokituksia on tehty kodinkoneille, erityisesti kylmälaitteille ja televisioillekin[21], mutta tietoteknisille laitteille vastaavia luokituksia ei ole vielä olemassa.

Esimerkiksi 4K-videon katsominen isosta televisiosta ja lataus langattomalla laajakaistalla mobiiliverkon kautta on energiankulutuksen kannalta huono ratkaisu, jos samaan kokemukseen päästään kännykällä tai kannettavalla tietokoneella, joka on kytketty langattomaan lähiverkkoon ja edel-

[21] ec.europa.eu/energy/eepf-labels/label-type/televisions_en

leen kiinteällä yhteydellä Internetiin. Jälkimmäisissä tapauksissa videon resoluutiokin luultavasti tippuisi automaattisesti, koska näyttö ei pysty hyödyntämään parempaa resoluutiota ja fiksusti toteutettu sovellus ottaisi tämän huomioon – säästäen sekä energiaa että tiedonsiirron kustannuksia.

Sovelluksen siirto päätelaitteelle tulee myös huomioida. Mikäli kyseessä on laitteelle asennettava sovellus, se siirretään kerran ja asennuksen jälkeen ensimmäisen kerran käynnistyessään se voi siirtää lisää tietoa verkosta. Jokainen käyttökerta on taas kevyempi tiedonsiirron kannalta.

Jos taas sovellus toimii webbisivulla, se siirretään pahimmassa tapauksessa uudestaan päätelaitteelle joka käyttökerta. Selaimet käyttävät välimuisteja ladattujen tiedostojen tallentamiseen päätelaitteelle vähentääkseen verkkokutsuja, mutta välimuistin koko ei ole loputon ja jossakin vaiheessa selain siivoaa vähemmän käytetyt tiedostot pois. Vastaavasti puhelimen käyttäjä tai puhelin automaattisesti poistaa vähän käyttämiään sovelluksia tallennustilan loppuessa ja lataa osan niistä myöhemmin takaisin.

Mobiili- ja työpöytäsovellukset voivat hallinnoida itse omia välimuistejaan, jolloin kehittäjillä säilyy tarkka kontrolli siirrettävistä asioista. Mikäli tallennettavaa tietoa on enemmän kuin välimuistille voidaan osoittaa tilaa, kehittäjien tulee miettiä järkevä välimuististrategia, jonka pohjalta tehdään tilaa uusille tiedostoille tarpeen vaatiessa. Strategian määrittää jälleen kerran sovelluksen arkkitehtuuri ja käyttötavat. Webbisivuilla on mahdollista tallentaa tietoa selaimen tarjoamaan paikalliseen tallennustilaan, mutta sitä hallinnoi sovelluksen lisäksi myös selain.

Laitteiden näytöt ovat myös merkittäviä energiankuluttajia. Käytännössä kännykän näyttö kuluttaa eniten energiaa käyttöskenaarioissa, joissa se on päällä. Ainoastaan kaikkein verkkointensiivisimmissä operaatioissa näyttö ei ole suurin kuluttaja[22]. Näyttöjen kulutus on melko suoraan riippuvaista kirkkausasetuksista. OLED-näytöissä kirkkaiden ja tummien pikselien suhde vaikuttaa myös kulutukseen, koska niissä ei ole erillistä taustavaloa. Mitä tummempi ruutu, sitä vähemmän kuluu energiaa.

[22] trustworthy.systems/publications/papers/Carroll%3Aphd.pdf

Laitteiden päivityssykli vaikuttaa merkittävästi laitteiden päästöihin. Laitevalmistajien mukaan noin 15-18 %:a laitteen kokonaispäästöistä tapahtuu käytön aikana ja loput liittyvät valmistukseen ja logistiikkaan. Esimerkiksi Delliltä löytyy kattavat selvitykset laitteiden valmistuksen, logistiikan, käytön ja käytöstä poistamisen päästöihin.[23]

## 5.4 Perusmallista monimutkaisempaan

Kuten aiemmin totesin, perusmallia voidaan – ja yleensä kannattaakin – monimutkaistaa lisäkomponenteilla, jotka on suunniteltu joko nopeuttamaan tai tehostamaan sovelluksen toimintaa. Tällaisia ratkaisuja ovat esimerkiksi:

- **Välimuistit**, joihin voidaan tallentaa joko valmiita tai puolivalmiita prosessoinnin tuloksia tulevien pyyntöjen nopeuttamiseksi. Välimuisti voi sijaita palvelun edessä (välityspalvelin eli proxy), jolloin se tallentaa verkkoliikenteen mukaisesti vastauksia pyyntöihin ja palvelee vastaaviin pyyntöihin suoraan, tai palvelun vieressä, jolloin palvelu tallentaa välimuistiin useasti tarvitsemiaan tuotoksia. Osassa järjestelmissä välimuisti voi olla kiinteä osa toteutusta, esimerkiksi tietokannoissa.

- **Content Delivery Network (CDN)**, jossa verkossa tarjolla olevista resursseista luodaan paikallinen kopio lähelle käyttäjää. CDN nopeuttaa resurssien latautumista ja mikäli resurssi luodaan lennossa, esimerkiksi webbisivu, se myös tehostaa sovelluksen toimintaa tarjoamalla jo aiemmin haetun version sivusta ja efektiivisesti poistaa sivun luomiseen käytettävän energian. Samoin niillä voidaan suojautua esimerkiksi DDOS-hyökkäyksiä vastaan. CDN toimii parhaiten tilanteissa, joissa käyttäjiä on paljon tai resurssit ovat kalliita luoda lennossa.

[23] www.dell.com/en-us/dt/corporate/social-impact/advancing-sustainability/sustainable-products-and-services/product-carbon-footprints.htm#tab0=0

- **Geografinen hajautus**, jossa palvelinkeskuksissa sijaitsevat ohjelmistot sijoitetaan useaan eri paikkaan. Tarkoituksena on optimoida tiedonsiirtoa ja lyhentää vasteaikoja. Vaaditaan tyypillisesti merkittävä määrä käyttäjiä tai vasteaikojen liiketoimintakriittisyyttä, että ratkaisu otetaan käyttöön.

  Mikäli sovellus ei ole täysin tilaton palvelinkeskusten suhteen, ratkaisu vaatii palvelinkeskuksissa sijaitsevien tietojen jonkun tasoista synkronointia tai käyttäjien ohjaamista "omaan" palvelinkeskukseensa. Nämä lisäävät energiankulutusta verrattuna toimintaan yksittäisessä palvelinkeskuksessa.

- **VPN-yhteydet** muuttavat käyttäjien siirtoteitä reitittämällä liikenteen päätelaitteesta esimerkiksi yrityksen sisäiseen verkkoon. Tämä sisäverkko voi kuitenkin olla maantieteellisesti hajautettu ja käyttää myös VPN-yhteyksiä eri osiensa välillä. Jokaisessa erillisessä VPN-yhteydessä tieto salataan ja puretaan, mikä lisää myös laskentatehon tarvetta.

- **Analytiikka**, jossa käyttäjän toimia seurataan joko webbisivuilla tai sovelluksessa. Tyypillisesti käytetään valmista palvelua ja sen tarjoamaa seurantakirjastoa. Ratkaisu kasvattaa sovelluksen kokoa ja luo päätelaitteen ja analytiikkapalvelun välillä eräänlaisen tiedonsiirron sivuvirran, joka voi olla tiheärytmisempi kuin palvelun oma tietoliikenne.

Kaikissa ratkaisuissa tulee puntaroida etuja ja haittoja, myös energiankulutuksen kannalta. Tyypillisesti näillä ratkaisuilla parannetaan loppukäyttäjän kokemusta, kuten vasteaikaa, tai kehittäjien ja muiden sovelluskehitysprojektiin liittyvien henkilöiden ymmärrystä käyttäjien toimista. Energiankulutusta ei mietitä lainkaan.

## 5.5 Toteutus, testaus ja tuotantoonvienti

Vaikka suurin osa ohjelmiston energiankulutuksesta syntyy käytön aikana, ohjelmistojen toteutuksen ja testauksen osuutta ei kannata jättää huomiotta. Erityisesti viime aikoina erilaiset tekoälyyn perustuvat

ohjelmointityökalut, kuten GitHub Copilot[24], ovat yleistyneet ja helposti moninkertaistavat ohjelmiston kehittämiseen liittyvän energiankulutuksen. Toki ne myös nopeuttavat ohjelmointia merkittävästi ja hyvässä lykyssä myös tuottavat laadukkaampaa koodia kuin ainakin vähäistä kokemusta omaavat tekijät.

Ohjelmointi ei sinänsä eroa muusta toimistotyöstä, molemmissa ollaan näytön ja näppäimistön välissä pitkälti koko työpäivä – ellei istuta kokouksissa – ja työkalujen energiankulutus ei ole merkittävän suuri.

Sen sijaan testauksessa voi käynnistyä moninkertaisesti suuremmat prosessit, yleensä vieläpä täysin automaattisesti. Kun ohjelmoija siirtää tekemänsä koodin versionhallintaan, käynnistetään tyypillisesti useampia erilaisia prosesseja, joilla varmistetaan tuotetun koodin laadukkuus. Koodi saatetaan tarkastaa erilaisilla analysaattoreilla ja niiden jälkeen ohjelmistosta rakennetaan toimiva versio, joka asennetaan testipalvelimelle ja sitä vastaan tehdään tietty ennalta sovittu määrä testejä. Mikäli tässä prosessissa havaitaan virhe, koodi heitetään takaisin kehittäjälle virheiden korjaamista varten.

Prosessi parantaa ohjelmiston laatua, koska mahdollisimman moni virhe pyritään havaitsemaan heti kättelyssä. Virheiden korjausten kustannukset kasvavat sitä mukaa, mitä myöhemmin virhe on havaittu. Kalleinta niiden korjaaminen on tuotantoonviedyissä ohjelmistoissa. Eräiden arvioiden mukaan kustannus on noin 30-kertainen suunnittelun aikaiseen bugin löytämiseen.[25]

Tämä testausprosessi on kuitenkin raskas ja se liipaistaan käyntiin jokaisen versionhallintaan viedyn muutoksen kohdalta. Jos muutos on vaikkapa kirjoitusvirheen korjaus, niin prosessi on täysin turha. Tosin nykyisten systeemien kyvykkyys erottaa kirjoitusvirheen korjaaminen

[24] github.com/features/copilot

[25] The exponential cost of fixing bugs, deepsource.com/blog/exponential-cost-of-fixing-bugs

uuden ominaisuuden teosta on edelleen valitettavan huono. Siksi onkin turvallisempaa ajaa aina kaikki testit uudestaan.

Mikäli sovellus käännetään erikseen useammalle – tai jopa monelle kymmenelle – kohdejärjestelmälle, automatisoitu järjestelmä voi olla hyvin monimutkainen ja raskas. Sovellus käännetään jokaisen muutoksen yhteydessä uudestaan jokaista kohdejärjestelmää kohden, kyseisen järjestelmän virtualisoitu ympäristö käynnistetään, sovellus asennetaan ja lopuksi suoritetaan testit. Tämä toistetaan jopa satoja kertoja päivässä useille kymmenille alustoille.

Testausjärjestelmien energiankulutusta, kuten juuri muutakaan IT:n energiankulusta, ei seurata ja siksi se jää katveeseen. Tämä tilanne olisi syytä muuttaa ja miettiä, että mitkä testit kannattaa aina ajaa jokaisen muutoksen yhteydessä ja mitkä taas esimerkiksi kerran vuorokaudessa tai jopa pitemmillä sykleillä. Samoin testien kohdistaminen vain muuttuneeseen osaan ohjelmistoa olisi järkevää, mutta tällaiset integraatiot ovat harvinaisia. Ja onhan se toki niinkin, että joskus muutos yhdessä paikassa ohjelmistoa rikkoo toiminnallisuuksia aivan jossakin muualla.

Tuotantoonviennit ovat myös automatisoituja moderneissa ympäristöissä. Näin vähennetään merkittävästi inhimillisiä virheitä ja koneen suorittama prosessi on nopeampi kuin ihmisten ohjaama ja tekemä. Näitten vientien taajuutta kannattaa kuitenkin miettiä. Osassa organisaatioissa vientejä tehdään useita kertoja päivässä ja toisissa taas viikoittain tai kuukausittain. Toki ohjelmistojen luonne vaikuttaa tähän, mutta liian tiuhaan tapahtuva vienti aiheuttaa helposti turhaa energiankulutusta.

# Tiivistettynä

1. Useimmat nykyaikaiset ohjelmistot noudattavat asiakas-palvelin -mallia, jossa palvelimet tarjoavat palveluita loppukäyttäjän laitteille. Mallin energiankulutus jaetaan karkeasti kolmeen osaan: ohjelmistot palvelinkeskuksissa, tiedonsiirto palvelinkeskusten ja laitteiden välillä sekä loppukäyttäjien laitteissa olevat sovellukset

2. Energiankäyttö palvelinkeskuksissa sisältää sovelluspalvelimet, tallennuksen ja tietokannat. Pilviympäristöjen ja virtualisoinnin käytöllä voidaan pienentää energiankulutusta. Palvelimen käytön ja kuormituksen optimointi on avain energiankulutuksen vähentämiseen.

3. Tiedonsiirron energiatehokkuus riippuu reitistä ja etäisyydestä. Verkon tyyppi, esimerkiksi mobiilidata tai valokuitu, sekä tiedon siirretty matka vaikuttavat energian kulutukseen. Tiedonsiirron minimoiminen vähentää energian käyttöä.

4. Loppukäyttäjän laitteiden energiankulutus vaihtelee laitetyyppien – esimerkiksi matkapuhelimet, työpöytäkoneet – ja käyttötilanteiden mukaan. Näytön kirkkaus, välimuistitus ja tiedonsiirtomenetelmät vaikuttavat kaikki kulutukseen.

5. Lisäkomponentit kuten välimuistit, sisällönjakeluverkot (CDN), maantieteellinen hajautus, VPN-yhteydet ja analytiikka voivat vaikuttaa merkittävästi energiankulutukseen.

# 6 Kadonneen tehokkuuden metsästäjät

Aikanaan tietokoneet olivat hyvin tehottomia ja rajoitettuja esimerkiksi muistin tai tallennustilan määrien suhteen. Ohjelmien tuli olla tehokkaita, koska muuten niitä ei olisi pystytty toteuttamaan lainkaan. Nykyään tehoa on käytössä suhteessa normaaliin käyttöön käytännössä rajattomasti ja siksipä sitä on helppo hukata. Ainoastaan erityiskäyttö, esimerkiksi tieteellinen laskenta, tekoälyn opettaminen tai suurten datamassojen käsittely, nostaa optimoinnin merkittäväksi osaksi sovelluksen kehittämistä.

Ohjelmistot ovat myös kerrostuneet. Aikoinaan kaikki koodi piti kirjoittaa itse ja pikkuhiljaa alkoi tulla ensimmäisiä kirjastoja yleisten toimien tekemiseen. Nykyään on löydettävissä kirjastoja tai valmiita rutiineja melkein kaikkien toimien suorittamiseen. Koska sovellukset ja niiden ajoympäristöt ovat kasvaneet ominaisuuksiltaan ja monimutkaisuudeltaan merkittävästi, sovellusten rakentaminen kerrosteisesti on järkevää laadun ja kehityksen tehokkuuden takaamiseksi.

Tämä kuitenkin johtaa helposti älylliseen laiskuuteen. On helpompaa ja edullisempaa valita valmis kirjasto kuin kirjoittaa rutiini itse, vaikka rutiini olisikin melko yksinkertainen. Toki kirjaston ratkaisu on todennäköisesti testattu ja siten luotettavampi, mutta valmiiden kirjastojen sisälle katsotaan hyvin harvoin. Mikäli ratkaisu on riittävän nopea – vaikkakin tehoton isossa mittakaavassa – niin mitään ongelmaa ei havaita.

Tässä ei nyt haeta sitä, että jokaisen sovelluskehittäjän pitäisi kirjoittaa kaikki käyttämänsä koodi aina alusta loppuun saakka yksin ollakseen

varma sen energiatehokkuudesta. Tämä on tehotonta sekä ohjelmistotuotannon että monesti myös energiatehokkuuden kannalta, koska paljon käytettyjen kirjastojen koodi optimoituu pikkuhiljaa käytännön kokemusten myötä.

Mikäli kirjaston korvaamiseksi kirjoitettu koodi alkaa olla satoja tai tuhansia rivejä pitkä, virheiden mahdollisuudet kasvavat merkittävästi. Samoin ongelmia tyypillisesti ilmenee, jos koodissa pitää tehdä monimutkaisia operaatioita tai siinä on merkittävä määrä mahdollisuuksia poikkeamiin normiprosessista. Tällaisten tilanteiden tulisi soittaa hälytyskelloja ja ratkaisuja olisi järkevää pohtia rauhassa, mahdollisuuksien mukaan jonkun kollegan kanssa.

Aina on kuitenkin hyvä tarkastella omia käytäntöjä ja miettiä, voisiko asian tehdä toisin ja tehokkaammin – vieläpä niin, että tämän tehokkaamman tavan saisi jakoon muiden sovelluskehittäjien iloksi.

## 6.1 Hukka perii

Vastaavia esimerkkejä löytyy eri puolilta ohjelmiston kehittämistä. Toisaalta jokainen sovellus on omanlaisensa yksilö omien vahvuuksiensa ja heikkouksiensa kanssa. Jotta asiaa voi lähestyä analyyttisemmin, sitä voidaan jäsentää lean-menetelmistä tutun hukan (waste) avulla.

Leanissa “hukalla tarkoitetaan ylimääräisiä, tuottamattomia toimintoja, jotka hidastavat prosessia tai tuottavat tarpeettomia kustannuksia.” [26]

Energiatehokkuuden kannalta hukka voidaan määritellä ylimääräisiksi, tuottamattomiksi toiminnoiksi, jotka kuluttavat tarpeettomasti energiaa. Kuten leanissakin, energiahukkia on erilaisia. Seuraavassa käsitellään hukkaa erityisesti yritysten kannalta, koska suurin osa ohjelmistoista on ainakin toisesta päästä yritysten hallussa.

Tässä käsittelyssä oletetaan, että sovellus on sinänsä toteutettu oikein. Jos sovellus tekee tuhat turhaa tietokantakyselyä väärin kirjoitetussa silmu-

[26] fi.wikipedia.org/wiki/Lean

kassa jokaisessa operaatiossaan, lasken sen ensisijaisesti ohjelmointivirheeksi enkä pelkäksi hukaksi – vaikka hukkaahan tuokin operaatio lopulta on. Hukka on enemmän ajattelemisen tai toteutuksen laiskuutta kuin varsinaisia virheitä ja hukan poistaminen on hyvin harvoin suoraviivaista.

Erilaisia hukkia on suuri määrä – alla oleva lista ei ole millään tavalla kokonainen tai lopullinen – ja niiden merkitys energiatehokkuuden kannalta vaihtelee merkittävästi ohjelmiston luonteesta, käyttötavasta ja sen ympärillä olevasta kokonaisarkkitehtuurista riippuen. Hukat on kirjoitettu oman ajatteluni mukaiseen merkitysjärjestykseen, eli suurimmat muutokset saadaan keskimäärin aikaiseksi valitsemalla listalta aiemmin esiintynyt hukka. Mutta kuten todettua, tämä on ohjelmistokohtaista ja listan olennaisin tavoite on tarjota erilaisia näkökulmia tehokkuuden parantamiseen.

### 6.1.1 Turhat ohjelmistot

Mikäli ohjelmisto on kokonaisuudessaan turha, kaikki sen käyttämä energia on tuhlausta. Toki ohjelmiston hyödyllisyyden tai turhuuden määrittely on hyvin subjektiivista. Tyypillisiä selkeitä turhia ohjelmistoja ovat jäänteet aiemmilta ajoilta. Esimerkiksi järjestelmä, joka tarkkailee käytöstä poistettua prosessia.

Toinen tyyppiesimerkki on kuluttajatietokoneiden mukana tulevat esiasennetut sovellukset, joista moni on mielestäni tärkeä ainoastaan laitteen tekijälle ja suoranaista hukkaa loppukäyttäjälle.

Mikäli ohjelmisto todetaan olevan kaikkinensa pelkkää hukkaa, se kannattaa ottaa välittömästi pois käytöstä ja säästää sen käyttämät resurssit planeetan hyödyksi.

Yksi tapa löytää turhat ohjelmistot on käydä talousosaston kanssa läpi kaikki ICT-toimittajat ja selvittää, mitä palveluja heiltä ostetaan ja mihin tarpeeseen. Mikäli ICT-arkkitehtuuria ei ole dokumentoitu ja ylläpidetty, niin tällainen projekti voi synnyttää selvää säästöä poistamalla turhia ratkaisuja ja säätämällä käytettyjen ratkaisujen kokoluokkaa vastaamaan todellista tarvetta.

### 6.1.2 Käyttö väärään tarkoitukseen

Ohjelmistot elävät ajassa, kuten kaikki muukin yrityksissä. Kun yrityksen tilanne muuttuu, ohjelmisto saattaa jäädä päivittämättä ja sen käyttö ei enää vastaa tarkoitustaan. Tai sitä ei alunperinkään osattu toteuttaa tai ostaa tarkoituksenmukaisesti.

Erityisesti valmisohjelmistoja ja SaaS-palveluja voidaan käyttää osittain tai jopa kokonaan eri tarkoitukseen kuin ne on alunperin suunniteltu. Tässä ei nyt tarkoiteta romaanin kirjoittamista PowerPointilla, vaan vaikkapa rekrytoinnin ohjaamista myyntiin suunnitellulla CRM:llä. Molemmat prosessit ovat samankaltaisia ja tuollainen ristiin käyttö voi onnistuakin ihan hyvin.

Hankaluuksia syntyy, jos sovellus ei tue kaikkia tuettavan prosessin vaiheita tai ominaisuuksia, taikka ei salli itsensä mukauttamista vastaamaan niitä. Tällöin voidaan joutua tilanteeseen, jossa joudutaan esimerkiksi tekemään jatkuvasti raskaita hakuja tiettyjen tietojen etsimiseen. Yleensä tämä näkyy myös käyttäjille sovelluksen tai sen tietyn toiminnon hitautena.

Tyypillisesti tällaista ratkaisua on lähdetty kokeilemaan ja huomattu sen toimivan riittävän hyvin. Sitten tarpeita on tarkennettu vaikkapa käytännön oppien kautta ja ratkaisun jäykkyydet nousevat haasteiksi tai esteiksi. Tässä vaiheessa sovellukseen on voitu jo tallentaa vuoden tai parin ajan tietoa, jota on vaikea siirtää parempaan järjestelmään. Samoin tämän paremman järjestelmän toteutus- tai hankintakustannukset voivat olla liian isot saavutettavaan hyötyyn nähden tai uponneiden kustannusten harha[27] haittaa järkiperäistä päätöksentekoa.

Vastaavasti ratkaisuun voi syntyä tyhjäkäyntiä sen muuttuessa pikkuhiljaa vuosien saatossa, sillä ohjelmiston eri muokkaukset muodostavat historiallisia kerrostumia sovellukseen. Mikäli kyse on jonkun yrityksen tuottamasta ratkaisusta, myös sovelluskehittäjät vaihtuvat muutosten välillä ja aiempia toteutuksia ei ole aikaa tai halua tutkia rauhassa, vaan niiden

[27] fi.wikipedia.org/wiki/Sitoutumisen_eskalaatio

päälle tai viereen toteutetaan uutta koodia ymmärtämättä kokonaisuutta. Tällöin sovelluksen toiminnallisuudet voivat sinänsä vastata tarkoitustaan, mutta niiden toteutus ei ole välttämättä optimaalinen.

Historiallisten kerrostumien poistaminen on hankalaa, koska ne eivät tyypillisesti ole toisistaan selkeästi irti olevia paloja, vaan samalla on sekä kirjoitettu uutta koodia että muokattu vanhaa toimimaan uuden kanssa yhteen. Sovelluksen refaktorointi kokonaan tai osittain ratkaisisi tämän ongelman, mutta se on kallista ja virhealtista. Silti on tilanteita, joissa se on kokonaisedullisin polku eteenpäin. Tarkka arviointi ja sen pohjalta tehtävät päätökset ovat avainasemassa refaktorointia pohdittaessa. Hyvä testikattavuus auttaa refaktoroinnissa mahdollisesti tapahtuvien virheiden havaitsemisessa.

### 6.1.3 Käyttäjien virheet

“Ihmiset tekevät virheitä, koneet automatisoivat ne.” Anon.

Sovellusten käyttäjille sattuu virheitä. Sormi tai hiiren kursori on väärässä paikassa, tulee kirjoitusvirheitä syöttökenttiin ja niin edespäin. Jokainen virhe aiheuttaa sarjan toimintoja, jotka ovat sovelluksen tavoitteen kannalta hukkaa. Samoin virheiden estämiseen käytetyt toiminnot ovat hukkaa, mutta sinänsä hyödyllisiä estäessään suurempien hukkien synnyn.

Mitä paremmin sovellus estää käyttäjää tekemästä virheitä, sen vähemmän sen tarvitsee tehdä turhia toimenpiteitä. Samoin mitä pienemmällä vaivalla – näyttöjen tai syötteiden määrä – käyttäjä saa tehtävänsä ratkottua, sen vähemmän koneenkin tarvitsee ponnistella. Nämä kysymykset liittyvät enemmän käyttökokemukseen ja -liittymään kuin koodin tehokkuuteen, eli sovelluskehittäjien lisäksi asian ratkomisessa tulee olla myös muita asiantuntijoita.

Käyttäjät ovat erilaisia lähtökohdiltaan ja heillä voi olla rajoitteita sovelluksen käyttöön liittyen. Lohdullista asiassa on, että vanheneminen tasaa ihmisten välisiä eroja huonontamalla kaikkien kognitiivisia ja kehollisia kykyjä. Näiden asioiden huomioiminen on nimeltään saavutettavuutta ja

se on ollut viime aikoina nosteessa EU:n saavutettavuusdirektiivin[28] myötä. Saavutettavuus on myös energiatehokkuuden kannalta järkevää, koska se vähentää virheitä helpottamalla sovelluksen käyttöä kaikkien kannalta – ei pelkästään erityisryhmien suhteen.

Kannattaa huomata, että virheiden poisto ei välttämättä tarkoita parempia testejä tai ohjetekstejä, vaan paras tulos saadaan poistamalla virheen mahdollisuus. Suurempi ja suorastaan zeniläinen kysymys on, että kuinka tämä käytännössä tehdään.

Virheiden mahdollisuuksien poisto tai pienentäminen on kokonaisvaltaista suunnittelua, jossa mietitään sovelluksen tarjoamaa tai avustamaa prosessia ja pyritään tekemään siitä mahdollisimman suoraviivainen ja aiempaan tietoon pohjautuva. Mitä vähemmän käyttäjältä vaaditaan vuorovaikutteisuutta sovelluksen kanssa, sen vähemmän on vaaraa inhimillisille virheille.

Toisaalta virheitä voidaan poistaa myös yksinkertaisilla käyttöliittymän muokkauksilla tai tekstimuutoksilla. Käyttöliittymän testaus on tässä avainasemassa, koska ohjelman hyvin tuntevat kehittäjät eivät ole paras taho pohtimaan, kuinka sovellusta todellisuudessa tullaan käyttämään. Samoin käyttäjäpalautteiden ja asiakaspalvelusta vastaavien ihmisten haastattelut avaavat silmiä todellisille haasteille. Ratkaisuja sieltä suunnasta on harvoin tarjolla, mutta jo ongelman määrittäminen helpottaa muutosten tekoa.

### 6.1.4 Väärä arkkitehtuuri

Sovelluksen arkkitehtuuri määrittää, mikä on helppoa ja mikä vaikeaa sovelluksen koodille. Aina arkkitehtuuri ei ole optimaalinen sovelluksen tehtävää ajatellen, vaan sovelluksen koodissa joudutaan “taistelemaan” arkkitehtuuria vastaan tai tekemään asiat turhan monimutkaisesti. Tämä lisää energiankulutuksen lisäksi myös ohjelmistossa esiintyvien virheiden todennäköisyyttä.

[28] eur-lex.europa.eu/legal-content/FI/TXT/HTML/?uri=CELEX:32016L2102&from=FI

Tyypillisin syy arkkitehtuurin aiheuttamaan hukkaan on sen vanheneminen sovelluksen käyttötarkoituksen tai -tavan muuttuessa vuosien varrella. Joskus valittu arkkitehtuuri on väärä heti alusta lähtien, mutta tämän pitäisi näkyä jo ensimmäinen version kehittämisen aikana ja siihen pitäisi reagoida. Sovelluksen käyttämä kirjasto tai sovelluskehys voi olla vastaavalla tavalla arkkitehtuuriltaan väärä tai hankala sovelluksen toiminnan kannalta, mikä johtaa erilaisiin sovittamistarpeisiin ja muihin energiaa syöviin hankaluuksiin.

Arkkitehtuurin vaihtaminen tarkoittaa käytännössä sovelluksen osittaista kirjoittamista uudelleen, eli se ei ole mitenkään pieni operaatio. Jos sovelluksen rakenne on modulaarinen tai osaratkaisut on muuten toteutettu itsenäisiksi, ne voidaan mahdollisesti ottaa käyttöön pienellä vaivalla uudessa arkkitehtuurissa. Mutta pahin skenaario voi olla koko sovelluksen kirjoittaminen uusiksi.

Arkkitehtuuria voidaan laventaa myös yrityksen kokonaisarkkitehtuuriksi, jossa suunnitellaan erilaisten liiketoimintaan liittyvien järjestelmien yhteistoimintaa ja yhteensopivuutta. Tämä voi olla vastaavalla tavalla rakennettu väärin suhteessa liiketoiminnan tavoitteisiin, aiheuttaen hukkaa tiedon käsittelyssä, siirrossa ja tallennuksessa. Tällaisen korkeamman tason arkkitehtuurin muuttaminen on vielä suurempi urakka kuin yksittäisen sovelluksen muutokset.

### 6.1.5 Tietomallit tuottamassa hukkaa

Sovellusten pääasiallinen tehtävä on käsitellä tietoa ja esittää sitä käyttäjälle jossakin ihmiselle ymmärrettävässä muodossa. Tietoa tallennetaan ja käsitellään erilaisissa tietomalleissa, jotka pyrkivät jäsentämään tietoa sovelluksen toiminnan ja ihmisen tarpeiden mukaisesti.

Aina tässä ei onnistuta, vaan tietomalli voi olla puutteellinen tai tehoton haluttuun tarkoitukseen. Puutteelliset tietomallit olivat suurehko ongelma, kun muisti- ja tallennustila oli rajallista ja tietomallit suunniteltiin useasti niiden vaatimien tavumäärien kautta. Mikäli jotakin tietoa ei ollut tarjolla, sitä ei joko pystytty käsittelemään lainkaan tai se survottiin johonkin tietomallin vähemmän tarpeelliseen kenttään. Näistä ongelmista on päästy

kapasiteetin kasvaessa ja nykyään tietomallin laajentaminen sovelluksen sisällä on melko suoraviivaista, ellei tehdä töitä sulautettujen ohjelmistojen parissa. Tosin jos tietomalli tulee taustajärjestelmistä tai on muuten integroitu osaksi isompaa kokonaisuutta, sen refaktorointi voi olla hankalaa.

Suurempana haasteena onkin valita oikea tietomalli tarpeen mukaan. Voi käydä niin, että sovelluksen haluttu käyttö onkin tietomallin rakenteen vastainen ja sovellus joutuu jatkuvasti kulkemaan tietorakenteitaan pitkin etsiessään tai yhdistäessään tietueita näytettäväksi käyttäjälle.

Tällaisen ongelman korjaaminen on yleensä valitettavan haastavaa, koska tietomalli yhdessä arkkitehtuurin kanssa määrittää koko sovelluksen sisäisen rakenteen. Tarvittavat muutokset ovat syvällisiä ja virheherkkiä. Useasti päädytään sietämään ongelmaa sen ratkaisemisen sijasta.

Tietomallihukan yksi alalaji on käytettyjen tietokantojen rakenteiden sopimattomuus. Vaikka tietokantamoottorit osaavatkin optimoida kyselyjä erinomaisesti, rakenteellisille ongelmille nekään eivät voi lopulta mitään. Tietokannan rakenteiden refaktorointi ei ole suoraviivaista, mutta erityisesti kovaa kuormaa palvelevilla ratkaisuilla vaiva maksaa itsensä takaisin.

Riippumatta tietokannan rakenteen muokattavuudesta, on syytä varmistaa oikeiden indeksien olemassaolo. Indeksit eivät missään nimessä ratkaise kaikkia ongelmia, mutta niitä kannattaa kuitenkin käyttää.

Tietokantamoottoreissa on myös eroja, esimerkiksi relaatio- ja dokumenttitietokannoilla voidaan ratkaista tehokkaasti erilaisia ongelmia ja vastaavasti ne voivat olla hyvin tehottomia, jos niitä käytetään tarkoitustaan vastaan.

### 6.1.6 Turha tieto

Hyvin harvasta sovelluksesta poistetaan aktiivisesti tietoa. Toki on selkeitä tapauksia, joissa käyttäjän tiedot poistetaan käytön päätyttyä tai GDPR:n[29] pakottaessa, mutta ohjelmistot tykkäävät pääsääntöisesti hillota kaikenlaista tietoa – ja aivan turhaan.

Tietomallit suunnitellaan yleensä tiedon eheyden ja hakemisen kannalta. Poistamista ei välttämättä mietitä, koska tietoja saatetaan tarvita vuosia ja joskus jopa ikuisuuskin. Mutta kaikki historiatieto ei ole yhtä arvokasta ja on hyvä katsoa tietomallia myös poistamisen näkökulmasta. Esimerkiksi kymmenen vuotta vanhojen tehtävälistojen säilyttämisessä ei luultavasti ole mitään järkeä, mutta eiväthän ne palvelusta itsestään poistu ja käyttäjiin ei voi luottaa, että he poistaisivat kaiken tarpeettoman.

Sovellusten suunnitteluvaiheessa suosittelen suunnittelemaan tietomallin sallimaan tietojen osittaisen poiston. Tämä on fiksua sekä GDPR:n että energiatehokkuuden kannalta. Turha tieto nimittäin kertautuu: se näkyy pidentyneissä käsittelyoperaatioissa tietokannoissa, levytilan käytössä, varmuuskopioinnin kestossa ja kopioiden koossa, mahdollisen hakukoneen erillisessä tietokannassa ja tämän varmuuskopiossa, jos sellainenkin otetaan. Lista on pitkä ja kumulatiivinen.

Vanhempien sovellusten tietovarastojen sisältöjä olisi hyvä käydä katselmoimassa aika ajoin. Sieltä voi löytyä täysin turhaa tietoa, jota ei esitetä tai käytetä missään, mutta tietokannassa se olla möllöttää hukkana. Tai sitten sitä näytetään ja käsitellään, mutta käyttäjät eivät niitä mihinkään tarvitse – ja hukka tuplaantuu.

### 6.1.7 Optimoimaton tieto

Nykyaikaiset sovellukset siirtävät paljon tietoa sekä sisäisesti että muiden sovellusten kanssa. Koska valtaosa sovelluksista toimii palvelin–päätelaite -arkkitehtuurilla, tiedonsiirto on olennainen osa sovelluksen toimintaa.

---

[29] General Data Protection Regulation, en.wikipedia.org/wiki/General_Data_Protection_Regulation

Kuten aiemmin todettiin, tiedonsiirrolla on energiakustannus, joka perustuu siirrettyyn tavumäärään. Periaatteessa mitä vähemmän tietoa siirretään, sitä vähemmän kuluu energiaa.

Siirretyn tiedon määrää voi kontrolloida säätämällä millaista tietoa siirretään ja kuinka usein sitä siirretään eli muokataan siirron taajuutta. Koska tiedonsiirto on sovelluksen kehittäjän kannalta ilmaista tai hyvin halpaa – ellei mennä valtaviin tieto- tai käyttäjämääriin – siihen harvemmin kiinnitetään huomiota. Tyypillisin tilanne on, että käyttäjät valittavat sovelluksen hitaudesta tai tiedonsiirtolasku pääsee yllättämään sovelluksen omistajan.

Tyypillisesti siirretään kahdenlaista tietoa:

1. Sovellusten osien keskinäistä viestintää, jota määrittää osapuolten välinen protokolla.

2. Erilaisia tiedostoja tai striimejä, kuten kuvia tai videoita päätelaitteelle esitettäväksi tai liitetiedostoja palvelimelle talteen.

Ensimmäisen kategorian datamääriin on hankalampi vaikuttaa, koska esimerkiksi sovelluksen ja tietokannan välinen keskustelu käydään valmiita protokollia käyttäen. Toki karsimalla haettua tai lähetettyä tietoa siirtomääriin voidaan vaikuttaa, mutta tämä ei aina ole mahdollista.

Toiseen kategoriaan voidaan vaikuttaa käyttökokemussuunnittelulla vähentämällä esityskertoja tai korvaamalla esitystapa vähemmän kuluttavalla tavalla, kuten teknisesti vaihtamalla käytetty tiedostomuoto tehokkaampaan ja kuvien ja videoiden suhteen pienentämällä resoluutiota tai värien määrää.

Tiedostojen tehokkaampi välimuistitus päätelaitteessa tai niiden siirtäminen osana sovelluksen asentamista voi vähentää tarvetta tiedonsiirrolle. Välimuistien käyttö monimutkaistaa sovellusta ja tekee sen kehittämisestä kalliimpaa. Kaikissa tapauksissa tiedoston koon pienentäminen on järkevää ja kaikki mahdollinen kannattaa pakata.

Verkosta löytyy erilaisia palveluja, joilla tiedostoja voidaan optimoida. Esimerkiksi kuvien tai kirjasinten kokoon voidaan tehdä merkittäviä

pienennyksiä. Sustainable Web Design -kirjassa[30] esitellään ja luetellaan suuri joukko pienennysratkaisuja erilaisilla käytännön esimerkeillä ryydi-tettyinä. Esimerkiksi kirjasimille saadaan merkittäviä tilansäästöjä poistamalla turhia leikkauksia ja sovellukselle tai sivustolle tarpeettomia glyyfejä. Glyyfien poistossa pitää miettiä tulevaisuusturvallisuutta: vaikka tiettyä glyyfiä ei juuri nyt tarvita, niin tulevaisuudessa sille voi tulla tarve esimerkiksi ulkomaalaisen asiakkaan nimen esittämiseksi oikein.

### 6.1.8 Tiedonsiirto varmuuden vuoksi

Monesti on helpompaa siirtää tietoa useamman kerran varmuuden tai helppouden vuoksi. Tämä jää helposti huomaamatta, koska siirtoverkot ovat nopeita ja energiankulutus jää piiloon. Esimerkiksi tietojen synkronoinnissa voidaan siirtää joka kerta koko paketti palvelimelta päätelaitteelle pelkkien muutosten sijaan. Syynä tähän on vaikkapa pelko synkronoinnin epäonnistumisesta tai tiedon hitaasta hajoamisesta muutosten epätahtisuuden vuoksi. Ongelma ei ole helppo ja siksi sen kiertäminen on houkuttelevaa.

Ongelman ratkaisemiseksi on kuitenkin kehitetty erilaisia synkronointikirjastoja ja -protokollia. Näiden käyttöönotolla voidaan saavuttaa merkittäviäkin säästöjä tiedonsiirrossa. Käyttöönotossa tulee huomata, että sovelluksen kehittäminen ja ennen kaikkea virheiden etsiminen monimutkaistuu. Riskinä on myös tietojen jääminen vain osittaisesti eheään tilaan. Tätä riskiä kannattaa pohtia suhteessa saavutettuun hyötyyn.

Lisäksi on olemassa aidosti hajautettuja protokollia, jossa jokainen laite sekä lähettää että vastaanottaa tietoa, jolloin tietoa voidaan hakea lähemmästä sijainnista kuin palvelimelta.

### 6.1.9 Algoritmillinen tehottomuus

Lopulta päästään aiheeseen, joka on tyypillisesti ensimmäisenä kokeneella sovelluskehittäjällä mielessä, kun puhutaan tehokkaasta koodaamisesta. Valittujen algoritmien ja tietorakenteiden tehokkuus ja ennen kaikkea

[30] abookapart.com/products/sustainable-web-design

sopivuus tehtävään ovat perinteisesti olleet isoimpia kipupisteitä ohjelmistojen tehokkuutta arvioitaessa. Aiheesta opetetaan yliopistotason kursseilla ja algoritmien kanssa puurtaminen on kehittäjistä yleensä mukavaa.

Pitkään käytössä olleiden ohjelmointikielien ja kirjastojen algoritmit ja rakenteet ovat yleensä hyvin optimoituja ja siten energiatehokkaita, kunhan niitä käytetään oikeaan tarkoitukseensa. Sen sijaan itse kirjoitettujen tai väärin valittujen algoritmien tehokkuus voi olla hyvinkin heikkoa, jos kehittäjällä ei ole aikaa tai kykyä miettiä valittuja ratkaisuja tehokkuuden kannalta.

Tehokkaimmat algoritmit ja tietorakenteet ovat yleensä myös toiminnaltaan haastavampia ymmärtää kuin saman ongelman ratkaisevat perustason algoritmit. Kannattaa myös tiedostaa, että vanhat ja aikoinaan tehokkaat algoritmit eivät välttämättä ole tehokkaita moderneilla prosessoriarkkitehtuureilla.[31]

Tyypillisesti sovelluksessa on tiettyjä osia, joiden koodia suoritetaan merkittävästi enemmän kuin muita osia sovelluksesta. Tällaisia ovat esimerkiksi käyttäjien suorittamien tapahtumien käsittely (event loop), tietokantayhteyksiä ylläpitävä ORM-kirjasto tai käyttäjien sessioita valvova komponentti. Tällaiset solmukohdat on syytä tunnistaa esimerkiksi profiloinnilla ja jos sellaista ei ole käytössä, niin arkkitehtuurin tai tietovoiden analyysilla.

Solmukohtien optimointiin kannattaa panostaa enemmän ja jättää vähemmän käytetyt osat pienemmälle huomiolle. Joskus tällainen solmukohta on käytössä olevan kirjaston tai sovelluskehyksen koodin puolella ja sen optimointi voi olla mahdotonta. Tällöin pitää etsiä vain seuraava kohde, kunnes päädytään omaan koodiin ja voidaan aloittaa työt. Muuten kohteiden merkityksellisyys hiipuu liian pieneksi.

[31] jolynch.github.io/posts/use_fast_data_algorithms/

### 6.1.10 Harhaanjohtaminen

Käyttäjän tekemien virheiden lisäksi käyttäjiä johdetaan harhaan palveluissa tai tietyistä toiminnoista, tyypillisesti tilin sulkemisesta tai avun pyytämisestä, tehdään hankalaa. Harhaanjohtavat suunnittelumallit (dark patterns) ovat käyttöliittymäratkaisuja, joilla käyttäjää huijataan toimimaan oman etunsa vastaisesti, koska siitä on hyötyä palvelua tarjoavalle taholle.[32]

Aktiivisen harhaanjohtamisen, esimerkiksi yritetään saada käyttäjä ostamaan joku tietty palvelu kalliiseen hintaan, lisäksi on passiivista harhaanjohtamista. Tällöin käyttäjää pyritään estämään löytämästä yrityksen kannalta negatiivista toimintoa tai tekemällä negatiivisen toiminnon käyttämisestä hankalaa. Tyyppiesimerkki on tilauksen lopettamisen piilottaminen valikoihin tai vaikkapa tilausnumeron pyytäminen, jota ei pyydetä missään muussa kohtaa sovellusta.

Tällaiset harhaanjohtamiset aiheuttavat käyttäjän turhaa kuljeskelua palvelussa, joka lisää energiankulutusta. Samoin, mikäli käyttäjä ottaa vahingossa jonkun palvelun käyttöönsä ilman aietta tehdä näin, kyseisen palvelun tilaaminen ja peruminen on hukkaa.

Tällaiset suunnittelumallit ovat monin tavoin vahingollisia ja niiden käyttäminen on myös eettisesti arveluttavaa.

Harhaanjohtavia ratkaisuja voidaan toteuttaa myös vahingossa. Vaikka ajatus on hyvä, niin toteutus voi silti olla käyttäjän kannalta harhaanjohtava. Osa liiketoiminnan tavoitteista, vaikkapa sivustolla vietetty aika tai vierailtujen sivujen määrä, voivat ohjata sovelluskehitystä ottamaan tällaisia harhaanjohtavia tai muuten käyttäjän sijasta ainoastaan liiketoimintaa palvelevia malleja käyttöön.

Tällaiset mallit tulevat yleensä ilmi viimeistään käyttäjätutkimuksessa. Testihenkilöt mainitsevat kyllä, jos vaikkapa palvelusta uloskirjautuminen tai tilauksen päättäminen on tehty vaikeaksi.

[32] en.wikipedia.org/wiki/Dark_pattern

Samoin UX-kirjoittaminen, eli käyttökokemukseen liittyvien tekstien suunnittelu ja kirjoittaminen, auttaa vahingossa toteutettuja haitallisia malleja vastaan. Esimerkiksi virheviestien muotoiluun on hyvä kiinnittää huomiota, jottei käyttäjä yritä uudestaan ja uudestaan tehdä samaa mahdotonta tointa sovelluksella.

### 6.1.11 Liikaa koodia

Sovellukset kasataan nykyään suuresta joukosta kirjastoja, joiden tarjoamia toiminnallisuuksia, tietorakenteita ja algoritmeja yhdistellään sovelluksen logiikan mukaisesti. Kirjastot nojautuvat toisiin kirjastoihin, jotka nojautuvat taas seuraaviin. Pahimmillaan sovelluksen omaa koodia voi olla vain muutama prosentti verrattuna kirjastojen koodimääriin, ja sovellus voi käyttää vain muutamaa prosenttia kirjastojen tarjoamasta koodimäärästä.

Tämä muodostaa ongelman erityisesti sovelluksen siirrossa päätelaitteelle. Käytännössä kaikki sovellukset ladataan verkosta ja lopputuloksena sama kirjasto voi olla päätelaitteella kymmeniä kertoja eri sovellusten sisällä. Kuitenkin kirjaston koodista saatetaan käyttää vain murto-osaa. Tämän ratkaisemiseksi kannattaa käyttää paketointisovelluksia (bundler), jotka poistavat turhan koodin ja pakkaavat sovelluksen tiukkaan nippuun.

Mobiili- ja työpöytäsovellukset ladataan kerran asennettaessa ja mahdollisesti päivitysten yhteydessä. Webbisovellukset puolestaan ladataan merkittävästi useammin, riippuen välimuistien ja selainten asetuksista, jolloin ongelmakin on kertaluokkaa suurempi. Tätä haastetta voidaan pienentää säätämällä tiedostojen välimuistituspolitiikka järkeväksi.

Kirjaston koko ja riippuvuudet olisikin hyvä selvittää ennen päätöstä ottaa kirjasto käyttöön sovelluksessa. Mikäli koodia tarvitaan vain vähän, on syytä miettiä joko pienemmän kirjaston käyttämistä – jos sellainen on ylipäänsä vaihtoehtona – tai kirjoittaa tarvittava koodi itse. Avoimen koodin kirjastoista voi niiden lisenssien sallimissa rajoissa myös kopioida koodia osaksi omaa sovellustaan.

Mitä enemmän sovelluksessa on käytössä kirjastoja, sen useammin ne päivittyvät, mutta niiden päivitysaikataulut eivät ole tietenkään sidoksissa toisiinsa. Valitettavasti uudemmissa kielissä tai nopeasti kehittyvissä ympäristöissä, kuten esimerkiksi tällä hetkellä webin front-end -toteutuksissa, kirjastot elävät vielä paljon ja muutokset voivat rikkoa yhteensopivuuden. Jokainen lisätty kirjasto lisää myös ylläpitotyön määrää.

Toki asialla on kääntöpuolensa. Valmiin kirjaston koodi on tyypillisesti tuhansien projektien varmentamaa ja sillä on erillinen ylläpitäjä, joka päivittää koodia tietoturvaongelmien paljastuessa. Ylipäänsä kirjasto kehittyy ilman omaa vaivannäköä. Aina kehitys ei tietenkään ole oman sovelluksen kanssa myönteistä, muutokset voivat rikkoa sovelluksen toiminnan tai aiheuttaa isoja sisäisiä muutostarpeita.

### 6.1.12 Tehoton ohjelmointikieli

Ohjelmointikielten tehokkuudessa on merkittäviä eroja. Osa kielistä käännetään konekielisiksi, osa tulkattavaksi tavukoodiksi ja osa käännetään ajonaikaisesti tavukoodiksi ja tulkataan. Lisäksi kielten erilaiset painotukset ja sisäiset tietorakenteet vaikuttavat tehokkuuteen.

Kielten valinta ei kuitenkaan aina ole ohjelmoijien käsissä eikä sitä voida perustella pelkällä tehokkuudella – tai muuten kaikki ohjelmistot kehitettäisiin konekielellä, C:llä tai muilla vastaavilla rautaa lähellä olevilla kielillä. Kielen ominaisuudet auttavat ja suojelevat kehittäjää eri tavoin ja niiden ympärille syntyneeseen ekosysteemiin on aikojen saatossa päätynyt tietynlaisia ratkaisuja. Vaikka teoriassa kielet ovat yhdenvertaisia ollessaan Turing-täydellisiä[33], niin käytännössä näin ei ole. Kielten ominaisuudet, niitä ympäröivät ekosysteemit kirjastoineen ja komponentteineen sekä kehittäjien saatavuus vaikuttavat merkittävästi kielen käytettävyyteen halutussa ympäristössä ja kyseessä olevan ongelman ratkaisemisessa.

[33] Jos ohjelmointikieli on Turing-vahva tai Turing-täydellinen, se voi oletuksen mukaan simuloida minkä tahansa muun tietokoneen tai ohjelman toimintaa. fi.wikipedia.org/wiki/Turingin_kone

Portugalilaisen Beira Interiorin yliopiston tutkijat ovat tutkineet reilun parinkymmenen ohjelmointikielen tehokkuutta[34] kymmenen erilaisen testitehtävän avulla. Tehtävän tekemiseen kulunut aika ja käytetty muisti on mitattu ja niiden pohjalta kielet on laitettu tehokkuusjärjestykseen.

Testiasetelma ei ole täydellinen, koska rutiinien toteutukset eri kielten välillä eivät ole yhteismitallisia, vaan ne on poimittu eri kehittäjien tekemistä ratkaisuista. Tämän vuoksi tuloksia voi pitää kokonaisuudessaan korkeintaan suuntaa-antavina. Kuten olettaa saattaa, testeissä paljastui suuria eroja esimerkiksi käännettyjen ja tulkattujen kielten välille.

Tutkimuksen mukaan kolme energiatehokkainta kieltä ovat C, Rust ja C++. Yleisesti käytetyistä kielestä hyvin pärjäsivät myös Java, Swift, Go ja JavaScript. Tehottomimmat kielet olivat Perl, Python ja Ruby, lisäksi yleisimmin käytetyistä kielistä tehottomammassa päässä on Lua ja PHP. Ero tehokkaimman ja tehottomimman kielen välillä oli lähes 80-kertainen.

Testit olivat luonteeltaan matemaattisia, jolloin esimerkiksi IO-tehoa tai ekosysteemiin tuotettujen kirjastojen tehokkuutta ei selvitetty. Näillä on taas suuri merkitys modernissa sovelluskehityksessä, sillä se pohjautuu tiedon siirtämiseen ja sovellusten rakentamiseen valmiista kirjastoista.

Kielten ajoympäristöjen välillä on myös tehokkuuseroja. Samalle tulkattavalle tai tavukoodille käännettävälle kielelle voi olla erilaisia tulkkeja, joista jotkut tuottavat optimoidumpaa koodia kuin toiset. Kielen tehokkuus voi myös kasvaa – tai joskus huonontuakin – myös ajoympäristön uuden version myötä ilman yhtään koodimuutosta. Kannattaakin siis varmistaa, että sovellusta ajetaan mahdollisimman tehokkaassa ajoympäristössä.

Jokaiselle kielelle selvitettiin testien yhteydessä myös korrelaatio ajoajan ja energiankulutuksen välillä, joka oli kaikilla kielillä joko vahva tai erittäin vahva. Tästä voidaan päätellä, että toteutuskielestä riippumatta nopeammin toimiva sovellus on energiatehokkaampi kuin hitaampi. Muistin- ja energiankulutuksen tai ajoajan ja muistinkulutuksen välille vastaavia korrelaatioita ei syntynyt kuin muutamassa kielessä.

[34] haslab.github.io/SAFER/scp21.pdf

Koska sovelluksen kehittäjillä ei välttämättä ole mahdollisuutta valita kieltä täysin tai ollenkaan vapaasti, niin kielen vaihtamisen sijaan on syytä keskittyä optimoimaan nykyisellä kielellä tehtyä koodia. Toki on mahdollista kirjoittaa erityisen paljon käytetty osa järjestelmästä tehokkaammalla kielellä.

Kun järjestelmään lisätään uusia komponentteja, näiden kielivalinnassa on hyvä miettiä myös energiankulutusta. Yksisilmäinen ei kuitenkaan kannata olla. Soveltuvan tehokkaamman kielen osaajista voi olla pula, siitä uupuu tarvittavia kirjastoja tai nykyinen ja uusi kaavailtu kieli pelaavat huonosti yhteen.

Sovelluskehittäjien olisi kuitenkin hyvä osata useita kieliä, koska tämä laventaa projektiryhmän mahdollisuuksia valita kieli tehtävän mukaisesti ja usean kielen osaaminen helpottaa uusien kielien oppimista.

### 6.1.13 Hukka ohjelmiston käynnistyessä

Datakeskuksissa ja pilvipalveluissa on periaatteessa kahdenlaisia sovellusympäristöjä: sovellus käynnistyy kerran ja palvelee käyttäjiä prosessin sammuttamiseen saakka tai sovellus käynnistyy joka kerta palvelemaan käyttäjän pyyntöä.

Mikäli sovellus on ensimmäistä laatua, sen käynnistymisen yhteydessä on hyödyllistä valmistella sovellus pitkään ajoon ja tehdä ennakkoon toimia, jotka pitäisi tehdä joka tapauksessa jossakin välissä sovelluksen käyttöä. Tällaisia toimia ovat yleensä asetusten luku, tietokantayhteyksien ja muiden vastaavien avaaminen, yleisesti tarvittavien tietojen luku muistiin tai mahdollisten aliprosessien käynnistäminen. Jos esitoimet ovat erityisen raskaita tai niitä tarvitaan vain harvoissa tilanteissa, ne voi olla järkevää tehdä vasta varsinaisen tarpeen ilmetessä.

Jos taas sovellus on jälkimmäistä lajia – kuten PHP-sovellukset tai pilvifunktiot – käynnistyessä tulisi tehdä ainoastaan sellaisia toimia, jotka ovat välttämättömiä kaikkien pyyntöjen palvelemisessa. Sen jälkeen edettäisiin käyttäjän pyyntöjen palvelemisen vaatimusten mukaan, eli käynnistysprosessi olisi jyvitetty useampaan vaiheeseen. Ratkaisu ei sovi kaikkiin ti-

lanteisiin; sovelluksen arkkitehtuuri ei välttämättä salli käynnistyksen vaiheiden ripottelua muun koodin sisään tai se muuttuu liian sekavaksi jatkokehittämisen ja virheherkkyyden kannalta.

Tulee myös huomioida, että sovelluksen tilattomuus ja nopea käynnistyminen palvelemaan käyttäjien pyyntöjä voi mahdollistaa cold standby -hajautuksen hot standbyn sijaan, eli sovelluksesta ei pidetä vikatilannetta varten käynnissä olevaa kopiota odottamassa käytössä olevan sovelluksen vikaantumista, vaan uusi kopio käynnistetään vikatilanteessa. Näin säästetään palvelinresursseja muuhun toimintaan.

### 6.1.14 Turhat turhuudet

Nykyaikaisissa sovelluksissa ja webbisivuissa on ylimääräistä "silmäkarkkia", jolla sovelluksen käytöstä pyritään tekemään mukavampaa, inhimillisempää tai hauskempaa. Mikäli nämä toimet eivät helpota käyttäjien toimia tai vähennä virheitä, ne ovat todennäköisesti hukkaa. Tämä ei tietenkään tarkoita, että kaikki sovellukset tulisi laatia mahdollisimman riisutuiksi. Silmäkarkilla on paikkansa, mutta sitä kannattaa viljellä käyttäjäehtoisesti eikä ripotella kaikkiin mahdollisiin paikkoihin.

Ihmiset ovat edelleen metsästäjä-keräilijöiden vaistoilla varustettuja olentoja ja kaikki liikkuva kiinnittää huomiomme. Sovelluksia käytettäessä liike hyvin harvoin tarkoittaa hengenvaaraa, mutta silti silmämme hakeutuvat helposti kohti animointeja. Mikäli käyttäjän huomio halutaan kohdistaa vaikkapa muuttuneeseen kohtaan tai tarjolle tulleeseen uuteen toimintoon, animointi on järkevää. Toisaalta esimerkiksi suuressa osassa webbisivuista tällä huomionhakuisuudella ei ole mitään järkevää tarkoitusta, vaan sivulle ilmestyvät elementit animoidaan esille vain näyttävyyden takia.

Sovelluksen on tärkeää pystyä kertomaan käyttäjälleen, että käyttäjän painallus on rekisteröity tai hiiren kursori on alueella, jonka napsauttaminen merkitsee jotakin. Näin vähennetään kognitiivista kuormaa, epätietoisuutta ja virheitä, mutta ylimääräiset huomionhakijat on kuitenkin järkevää raivata kokonaan pois. Mikäli käyttäjän huomiota ei haluta kiin-

nittää animoitavaan asiaan, on animaatio hukkaa kahdella tavalla: se vie huomion olennaisesta ja kuluttaa turhaan energiaa.

Samaan kategoriaan voidaan lisätä suurin osa taustavideoista. Ylipäänsä videoiden käytössä tulisi olla maltillinen, koska ne ovat raskaita siirtää ja niiden informaatioarvo verrattuna datamäärään on köykäinen. Videoilla on aikansa ja paikkansa, sillä ne helpottavat asioiden ymmärtämistä, mutta useasti ne voitaisiin korvata animaatioilla – joka sekin on yleensä videota, mutta pakkautuu merkittävästi tehokkaammin – tai lyhyellä tekstillä.

Teknisessä mielessä osa animaatioiden ja muiden vastaavien viemästä tehosta voidaan säästää muokkaamalla sovelluksen tai webbisivun toimintaa. Mikäli sovellus tai selain on tausta-ajossa, eli päällimmäisenä on joku muu sovellus, prosessoinnin määrää tulisi keventää ja sammuttaa kaikki käyttäjälle näkyvien toimintojen päivittäminen. Kännyköissä käyttöjärjestelmä säätää myös sovellusten toimintaa tässä suhteessa.

Lisäksi sovelluksen tai webbisivun toimintoja voi olla käyttäjältä piilossa, ne odottavat vielä sivun alalaidassa tai käyttäjä on jo ohittanut kyseisen kohdan sivua vierittäessään. Tällöin voidaan esimerkiksi poistaa ladattuja kuvia ja videoita viemästä muistia ja tarvittaessa palauttaa ne käyttäjän palatessa. Tässä tulee tietenkin arvioida tai mahdollisuuksien mukaan seurata, kuinka moni käyttäjistä kulkee näkymissä takaperin, ettei poisto ja uudelleenlataus aiheuttaisi lisää kuormitusta.

Eri formaateissa on myös eroja, esimerkiksi GIF-animaatiot pyörivät selaimissa myös niiden ollessa poissa näkyvistä, kun taas CSS-pohjaiset animaatiot sammuvat automaattisesti niiden kadotessa käyttäjän nähtäviltä.[35]

## 6.2 Minimointi

Hukan lisäksi sovellusten energiatehokkuutta voi lähestyä myös minimoinnin kautta. Tällöin ei pyritä tunnistamaan asioita tai tapahtumia, joissa hukataan energiaa, vaan yritetään laatia mahdollisimman kompakti ratkaisu, joka kuitenkin täyttää sille esitetyt tarpeet.

[35] webkit.org/blog/8970/how-web-content-can-affect-power-usage/

Minimointi vaatii hyvää ennakkosuunnittelua. Jokainen tarve tulee kyseenalaistaa ja miettiä, voidaanko pärjätä ilman sitä. Mutta aidosti tärkeät tulee kuitenkin toteuttaa, sillä mikäli minimoinnin tuloksena syntyy vain puolikas ohjelmisto, jota joudutaan korjaamaan myöhemmin, todennäköisyys tehottomuuteen tai hukkaan kasvaa nopeasti. Sen sijaan hyvin suunniteltu minimaalinen ratkaisu on tyypillisesti elegantti toteuttaa ja suoraviivainen käyttää. Jotta minimoinnissa voidaan ylipäänsä onnistua, sekä liiketoiminnan että käyttäjien tarpeet tulee tuntea riittävän yksityiskohtaisella tasolla.

Erityisen hyvin minimointi toimii tilanteissa, joissa ratkaistavat haasteet ovat tarkasti määritettävissä. Lisäksi niiden vertailu lukujen avulla helpottaa suuruusluokkien ymmärtämistä ja päätöksiä ei tehdä omien ajatusten vinoumien ja pinttymien kautta.

Kun palvelun päätarkoitus on keskiössä, palvelusta tulee selkeä ja käyttäjäkokemus paranee. Samalla konversiot kasvavat ja käyttäjäpolut lyhenevät, mikä parantaa sekä liiketoiminnan tuloksia, energiatehokkuutta että käyttäjien kokemusta. Vaatii uskallusta irroittautua toiminnallisuuskeskeisyydestä, mutta tulokset ovat palkitsevia.

Tarpeiden karsimisen lisäksi kohderyhmän rajaaminen helpottaa minimointia, koska samalla rajataan pois erilaisia käyttöskenaarioita. Rajaus kannattaa tehdä rooleittain, eli esimerkiksi tarjotaanko palkanlaskijoille suunnatussa palvelussa näkymiä myös johdolle ja työntekijöille, vai hoidetaanko heidän tarpeet jotakin muuta kautta. Asiantuntijoille ja muille prokäyttäjille voidaan lisäksi laatia omia käyttöliittymiä palvelun pääfokuksen ympärille, jolloin ne eivät sotke peruskäyttäjien tarpeiden tyydyttämistä.

Erityisesti kuluttajapalveluissa on erittäin tärkeää pitää huoli, että kohderyhmän rajauksella ei kavenneta jonkun ihmisryhmän mahdollisuuksia käyttää palvelua, koska silloin saatetaan päätyä syrjintään. Esimerkiksi saavutettavuuden poistaminen sovelluksen vaatimuksista ei ole järkevää minimointia.

Minimoinnin etuna on sovelluksen yksinkertaistuminen, jolloin tarvitaan lyhyempiä käyttäjäpolkuja, vähemmän toimintoja ja niiden virhe-

tilanteiden varmistuksia, käyttöliittymä on selkeämpi ja vähemmän virheherkkä ja käyttäjän kognitiivinen kuorma jää pienemmäksi, sovelluksen tarjotessa rajoitetummin visuaalisia viestejä.

Kannattaa myös muistaa, että minimum viable product (MVP) ja minimointi ovat kaksi hyvin eri asiaa. MVP on ensimmäinen versio tai prototyyppi ratkaisusta, ja sillä testataan idean toimivuutta. Minimaalinen sovellus taas on kokonainen ratkaisu, jota ei ole tarkoitus laajentaa myöhemmin ilman merkittävää syytä.

MVP-ajattelussa on kuitenkin se etu, että sovellus toteutetaan vastaamaan oletettuja tärkeimpiä tavoitteita. Matkan aikana opitaan lisää markkinoista ja käyttäjistä ja näin pyritään tekemään mahdollisimman järkeviä ratkaisuja tiukan priorisoinnin perusteella. Mikäli tämä ei johda kerrostuneisuuteen tai muuhun hukkaan, on tuloksena napakasti tarpeet täyttävä ratkaisu, jossa ei ole ylimääräistä.

## Tiivistettynä

1. Nykyisen ohjelmistokehityksen käytössä olevat runsaat resurssit voivat johtaa optimoinnin laiminlyöntiin, mikä aiheuttaa tehottomuutta. Kehittäjien tulisi välttää vain oman työnsä tehostamista ja ottaa energiatehokkuus yhdeksi tavoitteekseen.

2. Ohjelmistojen tehottomuuden haastetta voidaan lähestyä leanista tutun hukan avulla. Näin huomio keskitetään energiankulutusta lisääviin, arvoa tuottamattomiin toimintoihin.

3. Oikean arkkitehtuurin, tietomallien, algoritmien ja ohjelmointikielten valitseminen sekä tiedonsiirron hallinta ja tarpeettomien elementtien karsinta vaikuttavat merkittävästi ohjelmiston energiankulutukseen.

**4** Käyttäjäkeskeinen suunnittelu ja saavutettavat käyttöliittymät minimoivat käyttäjien virheet ja niistä syntyvän hukan.

**5** Minimointi tarjoaa vaihtoehtoisen tavan rakentaa energiatehokkuutta– oleelliseen keskittyminen ja ylimääräisten toimintojen poistaminen yksinkertaistaa ohjelmistoa, jolloin käyttökokemus ja energiatehokkuus paranevat.

# 7 Ratkaisuja

Edellisessä kappaleessa lueteltiin suuri joukko erilaisia tehohukkia. Käsitellään niiden pohjalta erilaisia tapoja keventää ohjelmistojen tuottamaa ympäristökuormaa. Nämä ratkaisut yleensä parantavat ohjelmiston toimintaa usealla mittarilla mitattuna, koska energiankulutus ja ohjelmiston käyttämä aika korreloivat vahvasti. Käytetty aika taas näkyy sovelluksen käyttäjälle hitautena, joka on yksi suurimpia ärsytyksen aiheita sovelluksissa.

Samoin energiankulutuksen vähentäminen pienentää käyttöpalvelumaksuja. Muutokset voivat olla erittäin tuntuvia pilvipalveluissa, joissa laskutus perustuu toimintojen käyttömääriin, kellosykleihin tai siirrettävän datan määrään.

Yksikään näistä ratkaisuista ei ole hopealuoti, vaan niitä tulee soveltaa oman ohjelmiston kontekstissa. Ja mikäli nämä kaikki ovat jo tuttuja ja käytössä, niin onnittelen kyvystäsi tuottaa tehokkaita sovelluksia. Otan myös mielelläni vastaan uusia ideoita ekotehokkuuden kasvattamiseksi.

## 7.1 Tallennetun tiedon minimointi

Mitä vähemmän sovellus käsittelee ja tallentaa tietoa, sen vähemmän se myös sitä siirtää päätelaitteen ja palvelimen välillä.

- **Sovelluksen tietomallin minimointi** – poistetaan tarpeeton tai vanhentunut tieto, laaditaan prosessit tietojen helpoksi poistamiseksi, tallennetaan vain prosessin lopputulokset ja tuhotaan väl-

itulokset. Tallennetaan mahdollisuuksien mukaan tieto tarkoituksenmukaisessa muodossa, käyttäen järkevästi kielen ja tietovaraston tarjoamia tietotyyppejä.

- **Sovelluksen staattisten tietojen minimointi** – valitaan oikea formaatti ja poistetaan tarpeettomat tai käyttäjälle huomaamattomat asiat. Kuvissa ja videoissa oikean resoluution ja pakkauksen aggressiivisuuden säätö on olennaista asettaa oikein. Lisäksi on syytä miettiä, että voidaanko esitysmuoto vaihtaa toiseen – vaikkapa kuvatusta videosta paremmin pakkautuvaksi animaatioksi tai pariksi kuvaksi. Kaikki tämä säästö kumuloituu, koska prosessointi tarvitaan tehdä vain kerran. Osa teknisistä muutoksista voi vaatia käsitöitä. Kaikki ohjelmallisesti komennettavat toimet on hyvä automatisoida osaksi CI/CD-putkia.

- **Käyttäjien syötteen minimointi** – mikäli käyttäjä voi liittää omia kuviaan tai videoitaan palveluun, ne kannattaa rajata ja pienentää sopivaan kokoon, pakata järkevällä tasolla ja tallentaa tehokkaassa formaatissa. Tämä on järkevintä tehdä päätelaitteessa, jos vain mahdollista. Sovellukselle käyttäjältä tulevaa tietoa voi myös minimoida rajaamalla käyttäjän syötteiden kokoa, esimerkiksi videon pituutta.

- **Tiedon kylmävarastointi** – joskus tietoa ei voida poistaa, vaan se tulee olla saatavissa ainakin jollain vaivalla. Tällöin tiedon kylmävarastointipalvelut, esimerkiksi Amazon Glacier tai Google Cloudin Archive, voivat olla järkevämpi ratkaisu pitää tieto tallessa kuin tiedon pitäminen jatkuvasti saatavilla osana sovellusta.

- **Analytiikkatiedon minimointi ja poisto** – sovellukset ja sivustot tallettavat omasta toiminnastaan analytiikkatietoa, jota käytetään sekä palvelun tekniseksi parantamiseksi että käyttäjien tarpeiden ymmärtämiseksi. Sovelluksen virheettömyys ja liiketoiminnan jatkuvuus ovat molemmat tärkeitä, mutta on hyvä puntaroida mitä tietoa tarvitaan ja kuinka pitkäksi aikaa.

## 7.2 Siirretyn tiedon minimointi

Siirretty tieto on tiukasti kytköksissä tiedon minimointiin, eli mitä vähemmän käsitellään, sitä vähemmän siirretään. Mutta lisäksi on tiedon siirtämiseen liittyviä erityishuomioita:

- **Taajuuksien säätäminen** – selvitetään käyttäjäkokemuksen ja sovelluksen sisäisen tilan kannalta järkevin taajuus siirtää tietoa. Energiatehokkuuden kannalta mahdollisimman pitkät välit ilman kommunikointia ovat hyviä, mutta ne voivat häiritä käyttökokemusta.

- **Tiedon pakkaaminen** – mikäli formaatti ei sisällä itsessään pakkaamista, käytännössä kaikki tieto on järkevää pakata ja purkaa. Tekstimuotoinen tieto pakkautuu murto-osaan alkuperäisestä jo sangen tehokkaasti. Pakkaamisesta saadaan lisäetua, mikäli tietoa on yhdistelty alempana esitellyn kohdan mukaisesti.

- **Protokollan tai viestimuodon valinta** – otetaan energiatehokas protokolla tai viestimuoto käyttöön. Näissä on eroja, sillä osa on hyvinkin puheliaita ja toiset selviävät pienemmällä datamäärällä. Mikäli tieto on hyvin määrämuotoista tai sitä on paljon, voidaan pohtia myös binäärimuotoisen viestinnän käyttöä tekstimuotoisen sijaan. Muodosta toiseen muuttaminen on aina kustannus, eli on syytä suhteuttaa siirtomäärien pienenemisestä saadut hyödyt sovelluksen monimutkaistumiseen ja ajoajan kasvuun.

- **Esitystiedon siirron poisto** – palvelin voi tuottaa HTML-koodia, joka näytetään päätelaitteella joko selaimessa tai sovelluksen sisällä. Jos siirtoa tapahtuu paljon, voi olla järkevämpää siirtää pelkkä sisältö mahdollisimman napakassa muodossa ja tuottaa HTML-koodi vasta päätelaitteessa jonkun template-kirjaston avulla.

- **Muuttuneen tiedon siirto** – mikäli sama tieto on sekä palvelmella että sovelluksessa, tilan päivittämiseen riittää ainoastaan muuttuneen tiedon siirtäminen. Tähän löytyy valmiita kirjastoja, joita on syytä käyttää mahdollisuuksien mukaan. Muutos ei ole triviaali ja se voi vaikeuttaa ylläpitoa ja ongelmanselvitystä huomattavasti.

- **Muuttumattoman tiedon tunnistaminen** – mikäli tieto ei muutu tai muuttuu hyvin harvoin, se voidaan pitää päätelaitteessa tallessa ilman erääntymispäivää. Palvelimet voivat tarjoilla tällaiset tiedot erilaisista välimuistiratkaisuista sovelluspalvelimen sijaan. Tiedonsiirto voidaan toteuttaa myös siten, että palvelin kertoo sovellukselle muuttuneista tiedoista osana muuta viestintää ja sovellus hakee nämä tiedot vasta saatuaan kyseisen ohjeistuksen.

- **Tietojen tarkastaminen ennen lähetystä** – käyttäjän päätelaitteessa syötetyt tiedot on syytä tarkastaa ennen niiden lähettämistä palvelimelle, jolloin vähennetään virheviestien lähettämistä. Tulee muistaa, että päätelaitteessa tehtävä tarkastus ei poista tarvetta tarkastaa tiedot myös palvelimella. Jos tarkastus on raskasta ja siirrettävä tietomäärä vähäinen, voi olla kokonaisekologisesti järkevää tehdä tarkastus vain palvelimella.

- **Tietojen yhdistäminen siirtoa varten** – niputetaan useampi viesti samaan lähetykseen, esimerkiksi raportoidaan käyttäjäanalytiikan tiedot käyttäjän palvelimelle lähettämien tietojen mukana. Tämä vaatii, että viestit ovat helposti niputettavissa ja lopputulos pakkautuu paremmin kuin viestit erikseen.

- **HTTP-headerien minimointi** – suurin osa viestinnästä tapahtuu nykyään HTTP:n päällä. Valmiit kirjastot saattavat liittää automaattisesti erilaisia headereita pyyntöihin tai vastauksiin. Näiden katselmointi ja turhien poisto on järkevää energiankulutuksen vähentämiseksi.

- **HTTP-uudelleenohjausten vähentäminen** – resurssien siirtyessä verkossa uuteen paikkaan, käyttäjien elämän helpottamiseksi webbipalvelimille voidaan lisätä uudelleenohjauksia, jolloin HTTP-pyyntöön vastataan ohjaamalla tekemään uusi HTTP-pyyntö toiseen osoitteeseen. Tyypillisempiä ovat esimerkiksi WordPressin tiettyjen osoitteiden perään lisättävä kauttaviiva. Tällaiset ylimääräiset lenkit tulee poistaa mahdollisuuksien mukaan käyttämällä täsmälleen oikeaa osoitetta kutsussa.

- **Palvelinten välisen siirron minimointi** – tyypillisesti palvelimessa on osana tietokantapalvelin tai vastaava, johon käsiteltävä tieto tallennetaan. Tietoa haettaessa kannattaa minimoida kerralla haettava tietomäärä oikean kokoiseksi, jotta turhaa tietoa ei haettaisi ja toisaalta uutta tiedonhakukutsua ei tarvitsisi tehdä.

## 7.3 Koodin vähentäminen

Sovelluksen koodin vähentäminen johtaa sovelluksen koon pienenemiseen, jolloin se on kevyempi siirtää verkon yli ja sen käynnistyminen saattaa nopeutua ja tehostua.

- **Kuolleen koodin poisto** – mikäli sovelluksen tiettyä osaa ei enää käytetä, koodin voi yksiselitteisesti poistaa. Jos siihen tulee joskus tarvetta palata, voi vanhan koodin löytää versionhallinnasta. Jos taas versionhallinta ei ole käytössä, niin se on syytä ottaa käyttöön per heti. Versionhallinta vaatii sekä tallennustilaa että prosessointikapasiteettia, mutta vastaavasti parantaa merkittävästi ohjelmistokehitysprosessin luotettavuutta ja karsii virheitä tuotetusta ratkaisusta.

- **Kirjastojen määrän vähentäminen** – sovelluksen koodista valtaosa on luultavasti kolmannen osapuolten kirjastoissa. Osa näistä on pieniä ja tehokkaita, toiset taas kookkaita ja huonosti toteutettuja. Ulkopuolelta tätä on vaikea nähdä. Jos kirjastosta käytetään vain pientä nurkkaa, on järkevää pohtia mahdollisuutta ottaa tämä ominaisuus osaksi omaa sovellusta lisenssiehtojen niin salliessa ja poistaa kirjasto. Samalla tulee huomata, että korjausvastuu mahdollisista koodissa piilevistä ongelmista siirtyy itselle.

  Osasta kirjastoista on olemassa kevyempi versio, joka tekee esimerkiksi 90 % toiminnoista paljon pienemmällä koodimäärällä.

- **Sovelluksen ominaisuuksien vähentäminen** – jos joku ominaisuus ei ole laajalti käytössä tai suorastaan vanhentunut, on järkevää poistaa se tarjolta käyttöliittymästä ja sen jälkeen poistaa näin syntynyt kuollut koodi.

- **Sovelluksen kääntäminen eri ympäristöihin** – jos sovellus voidaan asentaa useisiin erilaisiin ympäristöihin, esimerkiksi käyttöjärjestelmän eri versioihin, sovellustiedostosta voi olla ympäristöspesifisiä osia tai sovelluksen koodi kahdelle eri arkkitehtuurille. Yhden yhtenäisen binäärin tarjoamisen sijasta voi olla järkevää kääntää sovellus erikseen eri ympäristöihin ja tarjota oikea versio sovelluksen lataamisen yhteydessä. Mobiilisovellusten suhteen sovelluskaupat tekevät tämän automaattisesti.

## 7.4 Sovelluksen tehokkuuden parantaminen

Koska sovelluksen ajoaika ja energiankulutus korreloivat vahvasti, kaikki sovelluksen tehokkuuden nostamiseksi tehdyt toimet parantavat myös sen ekologisuutta. Aiheesta on valtavasti kirjallisuutta ja siitä tehdään yliopistotason tutkimusta, joten alla on kosketettu vain pintaa.

- **Oikeiden algoritmien valinta** – algoritmeilla on eroja ja osa niistä sopii hyvin laajojen massojen käsittelyyn ja toiset taas ovat tehokkaita vain pienillä tietomäärillä. Koska sovelluskehittäjillä on yleensä kiire asiakas- tai aikataulupaineesta johtuen, ensimmäinen sopiva tai toimiva algoritmi saatetaan ottaa käyttöön miettimättä tehokkuutta sen enempää.

  Ongelma piiloutuu sangen tehokkaasti tyypillisten testidatasettien pienuuden ja kehittäjien koneiden tehokkuuden takia. Kannustan suosimaan kirjastoja ja algoritmeja, joiden tehokkuudesta on kirjoitettu ylipäänsä jotakin ja etsiä algoritmeja tehokkuuden kautta.

- **Oikeiden tietorakenteiden valinta** – tietorakenteet määrittävät vahvasti sovelluksen toimintaa. Osa on suunniteltu tehokkaiksi isoillekin tietomäärille ja osa murtuu jo pienemmälläkin määrällä. Lisäksi tietorakenteen pitää tukea sovelluksen toimintaa. Tehokaskaan tietorakenne ei pelasta, jos sovelluksen pitää jatkuvasti kulkea edestakaisin sitä pitkin etsiessään tietoa.

- **Optimointi oikeassa paikassa** – sovelluksessa on tyypillisesti muutamia solmukohtia, joissa vietetään suuri osa sovelluksen käyt-

tämästä ajasta ja energiasta. Nämä tulee tunnistaa ja niiden optimointiin käytetty aika on hyödyllistä. Sen sijaan optimointia ei kannata levittää tasaisesti koko sovelluksen alalle ja ennenaikainen (premature) optimointi on haitallista – eli sovelluksen toiminta täytyy tuntea syvällisesti ennen liiallista optimointia.

- **Refaktorointi** – mikäli sovelluksen toteutustapa ei vastaa tarpeita, se voidaan joutua kirjoittamaan osittain uudestaan eli refaktoroimaan. Tämä on työvoimaintensiivinen tehtävä, jossa myös syntyy helposti virheitä, eli tarpeiden pitää olla riittävän merkittävät. Refaktoroinnissa suosittelen miettimään kaikkia tämän kappaleen neuvoja ja soveltamaan niitä mahdollisuuksien mukaan. Kattavat testit tai tiukka tyypitys auttavat varmistamaan, että refaktoroitu toteutus toimisi vastaavasti kuin alkuperäinenkin.

- **Ajoympäristön vaihtaminen** – tulkattavien ja tavukoodiksi käännettävien kielten ajoympäristöissä voi olla merkittäviäkin tehokkuuseroja. Yleensä erot ovat eri ajoympäristöjen välillä, mutta myös versiopäivitys uudempaan voi tuottaa lisätehoa. On suositeltavaa seurata ajoympäristöjen kehitystä ja valita omalle sovellukselle parhaiten soveltuva ympäristö.

- **Toteutuskielen vaihtaminen** – joskus on järkevää vaihtaa osa sovelluksen toteutuksesta toiselle kielelle tehokkuussyistä. Tyypillisimpiä esimerkkejä ovat C:llä koodatut rutiinit, joita kutsutaan skriptikielillä tehdyistä sovelluksista. Ratkaisu ei ole suoraviivainen ja kaikkia kieliä ei voida helposti yhdistää keskenään, mutta erityisesti raskasta laskentaa vaativissa tilanteissa tämä on aito vaihtoehto. Koko sovelluksen toteuttaminen uudestaan eri kielellä yleensä taas kaatuu kustannuksiin.

- **Tausta-ajon huomiointi** – sekä työpöytä- että selainsovellukset ovat useasti taustalla käyttäjän keskittyessä muihin toimiin laitteellaan. Sekä käyttöjärjestelmät että selaimet muokkaavat automaattisesti toimintaansa tausta-ajoa varten kaventamalla tarjolla olevaa prosessoriaikaa priorisointia säätämällä ja muuttamalla esimerkiksi ajastettujen tehtävien ajoa. Tämän lisäksi asian voi huomioida

omassa koodissa, jos vain on saatavilla signaali tausta-ajoon siirtymisestä ja taas etualalle tulosta. Taustalla ollessa jätetään käyttöliittymä päivittämättä ja säädetään reaktioaikoja pidemmiksi, esimerkiksi haetaan tietoa taustajärjestelmistä normaalia hitaammin.

## 7.5 Ulkoisten ratkaisujen käyttö

Tehokkuuden parantamiseksi on mahdollista käyttää myös erilaisia ulkopuolisia palveluja, jotka toimivat esimerkiksi välimuisteina palvelimen ja käyttäjän päätelaitteen välillä. Tyypillisesti tällaiset ratkaisut toimivat sisällönhallintaan liittyvissä toimissa, mutta toki niitä voi käyttää muuallakin luovasti. Muitakin ulkoisia ratkaisuja on tarjolla ja niiden etuna on tiettyyn tehtävään optimoitu koodi ja joskus myös rauta.

- **Content Delivery Networkin käyttöönotto** – CDN tallentaa tietoa lähelle käyttäjää, tyypillisesti heillä on palvelimet eri operaattoreiden runkoverkoissa. CDN:n käyttö on järkevää erityisesti, kun palvelulla on paljon käyttäjiä, joille palvellaan samoja tietoja tai tiedostoja, tai kun palvelu on globaalisti hajautettu ja halutaan lyhentää latenssia. CDN tallentaa tiedot omille palvelimilleen ympäri maailman, eli tiedon määrä moninkertaistuu.

- **Välimuisti- ja kuormantasauspalvelimet** – on olemassa erilaisiin tarkoituksiin suunniteltuja väliaikaisen tai pysyvänkin tiedon nopean tallentamisen ja hakemisen ratkaisuja. Näillä voidaan esimerkiksi jakaa paljon prosessointitehoa vaativia välituloksia usean sovelluspalvelimen kesken. Tässäkin tapauksessa tulee punnita sovelluksen monimutkaistumisen ja välimuistissa olevien tietojen ajantasaisena pitämisen tuomia haasteita.

- **Ulkoiset salausratkaisut** – mikäli järjestelmällä on laaja käyttö, ulkoisen SSL-kiihdyttimen käyttöönotto voi olla järkevää. Tämä voi vaatia uusien laitteiden käyttöönottoa, mutta tällaiseen käyttöön suunnitellussa raudassa voi olla merkittävästi tehokkaampi toteutus tarvittaville salausalgoritmeille. Vastaavasti voi olla järkevää käyttää jo käytössä olevaa VPN-ratkaisua sisäisten sovellusten viestinnän

turvaamisessa sen sijaan, että käytetään sovelluksen salaamia yhteyksiä.

## 7.6 Muita ratkaisuja

Maailma on siirtymässä uusiutuvien sähköntuotantotapojen yleistymisen vuoksi suurempaan volatiliteettiin energian kustannuksissa. Tämä yhdistettynä Eurooppaa ravistelevaan energiakriisiin Ukrainan sodan takia on nostanut energiankulutuksen ajankohdan merkitykselliseksi. Volatiliteetti tuskin poistuu muutaman vuoden sisällä, koska energian varastointi on vielä lapsenkengissä.

Nyrkkisääntönä on, että kun energia on halpaa, se on myös uusiutuvaa. Mikäli ohjelmistossa on tarvetta tehdä laskentaa, joka ei ole sidottuna tiettyyn hetkeen, kannattaa sitä ohjata edullisemman energian hetkiin. Näin sekä kulutetaan mahdollisimman puhdasta energiaa että tasataan energian hinnan piikkejä, millä on vaikutusta esimerkiksi kaikkein haavoittuvimmassa asemassa olevien ihmisten elämään.

Tällaisia ratkaisuja on jo olemassa ja GitHubista löytyy esimerkiksi Carbon Aware SDK[36], jolla oma sovellusten toimintaa voidaan sovittaa sähkön hinnan mukaisesti.

Myös kehityspalvelimien ja -ympäristöjen energiankulutusta voi tarkastella kriittisesti. Voidaanko esimerkiksi staging- ja testipalvelimet sammuttaa yöajaksi ja viikonlopuiksi? Ovatko ne kooltaan järkeviä vai ovatko ne käyttöön nähden liian tehokkaita ja sitä myöten energiaa enemmän kuluttavia? Staging- ja testiympäristöjen virtualisoinneilla ja kontituksilla voidaan käyttää laitteita mahdollisimman optimaalisesti.

[36] github.com/Green-Software-Foundation/carbon-aware-sdk

## Tiivistettynä

1. Vähennä sovelluksen tietojen käsittelyä, tallennusta ja siirtoa optimoimalla tietomalleja, staattisia tiedostoja ja käyttäjän syötteitä.

2. Vähennä tiedonsiirtoa optimoimalla tiedonsiirron taajuuksia, pakkaamalla dataa, valitsemalla energiatehokkaat protokollat ja rajaamalla HTTP-headereita sekä uudelleenohjauksia.

3. Pienennä sovelluksen kokoa poistamalla tarpeeton koodi, vähentämällä kirjastojen käyttöä ja poistamalla turhia tai vähän käytettyjä toimintoja.

4. Valitse sopivat algoritmit ja tietorakenteet, optimoi kriittiset osat sovelluksesta, refaktoroi tarvittaessa ja harkitse ajoympäristön tai toteutuskielen vaihtamista.

5. Arvioi kriittisesti kehityspalvelimien energiankulutusta, optimoi testaus- ja tuotantoympäristöjä sekä harkitse virtualisointia ja konttiteknologiaa resurssien optimaalista käyttöä varten.

# 8 Erityiset ratkaisut

Perussovelluskoodin lisäksi nykyaikana on käytössä useita energiaa merkittäviä määriä kuluttavia ratkaisuja, kuten tekoäly ja lohkoketjut. Nämä on syytä käsitellä erikseen, koska niiden energiankulutus poikkeaa perinteisten sovellusten koodin energiankulutuksesta. Pahimmillaan tällaisen ratkaisun käyttö voi räjäyttää kulutuksen. Toisaalta niiden käytöstä on myös kiistatonta hyötyä. Olennaista onkin tunnistaa, miten ja miksi energiaa kuluu ja voiko siihen puuttua.

Tässä luvussa esitellään muutamia yleisessä käytössä olevia ratkaisuja, joiden energiajalanjälki voi olla merkittävä.

## 8.1 Tekoäly

Tekoälyratkaisut tulivat suuren yleisön tietoon vuoden 2022 aikana, erityisesti ChatGPT:n ja Dall-E:n helpottaessa tekoälyn kokeilemista ja käyttämistä ilman erityistä ohjelmointiosaamista. Vastaavasti odotukset tekoälyn liiketoimintavaikutuksista kasvoivat eksponentiaalisesti. Esimerkiksi Gartner ennustaa 40% yritysten liiketoimintasovelluksista (“enterprise applications”) sisältävän generatiivisia tekoälytoiminnallisuuksia jo vuoden 2024 kuluessa. Vastaavasti he ennustavat, että 15% sovelluksista on tekoälyn kirjoittamia vuonna 2027.[37]

[37] Gartner Experts Answer the Top Generative AI Questions for Your Enterprise, www.gartner.com/en/topics/generative-ai

Tässä kappaleessa niputetaan erilaiset tekoälyt, koneoppiminen ja muut vastaavat termit käsitteen tekoäly alle.

Tekoäly on luonteeltaan energiaintensiivistä laskentaa ja sen energiankulutus voi olla suurta. Tekoälyn energiankulutuksesta on viime aikoina kirjoitettu enemmänkin, mutta tyypilliseen ohjelmistojen energiankulutuksen käsittelyn tapaan tarjolla ei ole juurikaan dataa – koska tekoälyä kehittävät yritykset eivät kerro lukujajulkisuuteen.[38] Tekoälysovelluksia ajetaan myös pitkälti pilvipalveluina, jolloin todellinen energiankulutus piiloutuu palvelun laskutuksen sisään.

Esimerkiksi Googlen sähkönkulutuksesta arvioidaan noin 10-15% johtuvan tekoälyn käytöstä.[39] Vuonna 2021 Googlen sähkönkulutus oli 18,3 terawattituntia, jolloin tekoälyn kulutus olisi jotakin 1,8 ja 2,7 TWh välillä. Pelkkä vaihteluväli 0,9 TWh on lukuna merkittävä, se vastaa Olkiluodon ydinvoimalan kolmosyksikön kuukauden tuotantoa.[40] Kolmosyksikkö ei myöskään yksin pysty kattamaan Googlen tekoälyn sähkönkulutusta, vuosituotannon ollessa noin 12 TWh.

Kuten esimerkeistä havaitaan, kulutuslukemat ovat karkeita arvioita ja vaihteluvälit ovat suuria. Tämä ei saa kuitenkaan estää todellisen kulutuksen arviointia ja minimointia.

### 8.1.1 Energiankulutus

Tekoälyn energiankulutus voidaan jakaa kolmeen osaan:

---

[38] As the AI industry booms, what toll will it take on the environment?, www.theguardian.com/technology/2023/jun/08/artificial-intelligence-industry-boom-environment-toll

[39] Artificial Intelligence Is Booming—So Is Its Carbon Footprint, www.bloomberg.com/news/articles/2023-03-09/how-much-energy-do-ai-and-chatgpt-use-no-one-knows-for-sure

[40] www.tvo.fi/tuotanto/laitosyksikot/ol3.html

- **Opetusmateriaalin koosto ja jäsentely.** Tekoälyä ei voi opettaa syöttämällä sille mitä tahansa. Tietokoneaikakauden "roskaa sisään, roskaa ulos" -periaate on voimassaan myös tekoälyn opetuksessa. Joten opetusmateriaali täytyy koostaa, sitä täytyy jäsentää ja se täytyy siivota. Koska opetusmateriaalia tarvitaan valtavasti, tämäkin työ tulee automatisoida, mikä puolestaan kuluttaa energiaa.

- **Tekoälyn opettamiseen kuluva energia.** Neuroverkkopohjainen tekoäly muodostaa verkon kytkennät opetusmateriaalin perusteella. Opetusmateriaalin määrä riippuu tekoälyn toteutusteknologiasta ja halutusta osaamisalueesta sekä tarkkuustasosta. Samoin opetusmateriaalin muoto vaikuttaa opettamisen energiankulutukseen, esimerkiksi video- tai kuvamuotoisen materiaalin läpikäynti vie enemmän tehoa per saatu informaatio kuin tekstimuotoisen materiaalin käsittely.

  Esimerkiksi GPT-3 -mallin, jossa on 175 miljardia parametria, opetuksen arvioidaan tutkijoiden mukaan kuluttaneen 1 287 MWh ja tuottaneen täten 552 tonnia hiilidioksidipäästöjä.[41] Samassa artikkelissa myös todetaan, että tekoälyn energiankulutus voi vaihdella sata-, jopa tuhatkertaisesti riippuen käytetystä infrasta. On myös huomattava, että opetusdatan tai tuotetun tekoälymallin koko ei määritä energiankulutusta, vaan eri tekoälymallit oppivat eri energiatehokkuuksilla.[42]

  Code Carbon on mitannut suosittujen luonnollisen kielen tuottamiseen ja konenäköön kehitettyjen mallien oppimisen energiankulutusta.[43] Erot ovat merkittäviä. Valitettavasti yhteismitallista op-

[41] Carbon Emissions and Large Neural Network Training, arxiv.org/abs/2104.10350

[42] A Computer Scientist Breaks Down Generative AI's Hefty Carbon Footprint, www.scientificamerican.com/article/a-computer-scientist-breaks-down-generative-ais-hefty-carbon-footprint/

[43] mlco2.github.io/codecarbon/model_examples.html

pimisen energiankulutustietoa ei ole saatavilla. Toivottavasti tähän asiaan saadaan korjaus lähitulevaisuudessa.

Opetuksen kulutusta miettiessä tulee myös huomioida, että opetusdata vanhenee ja tekoälyä tulee opettaa uudestaan aika ajoin – muuten se jäisi pikkuhiljaa vanhentuneen tiedon varaan ja menettäisi merkityksensä, eli todellisuudessa näemme vain jäävuoren huipun energiankulutuksen osalta tässä hetkessä

- **Tekoälyn käytössä kuluva energia.** Kun tekoäly laitetaan tekemään tehtäviä, esimerkiksi olemaan vuorovaikutuksessa ihmisten kanssa tai tutkimaan röntgenkuvia, neuroverkon käyttö kuluttaa energiaa. Neuroverkko toimii sinänsä samoin kuin mikä tahansa muukin ohjelmakoodi, eli tietoa siirretään muistista prosessille käsiteltäväksi ja takaisin. Verkon koko ja kompleksisuus johtavat siihen, että tätä käsittelyä ja tiedonsiirtoa tapahtuu erittäin paljon.

  Esimerkiksi ChatGPT:n yksittäisen kyselyn on arvioitu kuluttavan energiaa 1,7–2,6 Wh.[44] Virallista tietoa päivittäisistä kyselymääristä ei ole saatavilla. Useampi sivusto on arvioinut kyselyjen päivittäiseksi määräksi kymmenen miljoonaa kappaletta, jolloin päivittäinen sähkönkulutus olisi vastaavasti 17–26 MWh.

  Lisäksi käytönaikaiseen energiankulutukseen on syytä laskea tekoälymallin lataaminen palvelimen muistiin ja muut käyttöönottoon liittyvät valmistelut. Esimerkiksi ChatGPT:n GPT-3 -mallin kooksi arvioidaan noin 800Gt[45], joten sen lataaminen ei ole merkityksetöntä energiankulutuksen näkökulmasta.

Yhteenvetona voidaan todeta, että tekoälyratkaisut kuluttavat helposti paljon energiaa, tämä kulutus jää helposti piiloon pilvipalveluissa ja ratkaisuissa on merkittäviä eroja energiankulutuksen suhteen. Tämä puolestaan

[44] ChatGPT's energy use per query, towardsdatascience.com/chatgpts-energy-use-per-query-9383b8654487

[45] en.wikipedia.org/wiki/GPT-3

johtaa sovelluskehittäjien kasvavaan vastuuseen tekoälyn käytön ja ratkaisun valinnan tiimoilta.

### 8.1.2 Suositukset

Tekoälyn käyttö voi suurentaa sovelluksen energiankäyttöä useilla magnitudeilla, eli sen kanssa kannattaa olla tarkkana. Suosittelen pohtimaan seuraavia kysymyksiä:

- **Tarvitaanko tekoälyä laisinkaan?** Moni ongelma on tällä hetkellä muodikasta ratkaista tekoälyn avulla, vaikka se voisi ratketa heuristisilla tai tilastollisilla menetelmillä. Toki tämä ei ole vastaavalla tavalla mediaseksikästä ja voi esimerkiksi pienentää kasvuyhtiöiden valuaatioita.
- **Kuinka tekoälyn käyttö rajataan?** Jos tekoälyä kuitenkin syystä tai toisesta tarvitaan, niin kannattaa pohtia kuinka sen käyttö kannattaa rajata. Se on kuitenkin vain työkalu, ei itsetarkoitus. Tässä on haettava tasapainoa liiketoiminnan tarpeiden, laskennan oikeellisuuden tai todennettavuuden sekä energiankulutuksen suhteen.
- **Mikä tekoälymalli on oikea tarpeeseen? Kuinka se hankitaan?** Tekoälymallit soveltuvat erilaisiin tarpeisiin ja malli tulee valita oikean tarpeen pohjalta. Kannattaa valita jo käytössä oleva malli ja jos kyseessä on pilvipohjainen ratkaisu, pyrkiä käyttämään sellaista maantieteellistä sijaintia, jossa sähkö on mahdollisimman puhtaasti tuotettua.
- **Kuinka malli opetetaan? Mistä saadaan opetusmateriaali ja kuinka se valmistellaan?** Opetusmateriaalin tulee vastata liiketoiminnan tarpeita, mahdolliset vinoutumat tulee huomioida ja sen luomisen tulisi olla mahdollisimman energiatehokasta. Jos tekoäly tulee sisäiseen käyttöön tai ihminen ei voi olla sen kanssa lainkaan vuorovaikutuksessa, mahdollisia väärinkäytöksiä on luultavasti vähemmän. Tällöin esimerkiksi tekoälyn tuottamaan haitalliseen sisältöön voi suhtautua eri tavoin kuin täysin avoimessa ratkaisussa.

- **Miten malli konfiguroidaan?** Tekoälymallit sisältävät valtavasti niiden toimintaa muokkaavia parametreja, joista osalla on merkittäviä vaikutuksia energiankulutukseen. Aina kaikkein raskain laskenta ei tuota parasta mahdollista lopputulosta. Oikean tasapainon löytäminen vaatii kokeiluja.

Lopuksi kehotan kaikkia tekoälyn kanssa toimivia jakamaan omia kokemuksiaan ratkaisujen käytöstä ja energiankulutuksesta. Näistä tiedoista on tällä hetkellä huutava pula.

## 8.2 Lohkoketjut ja kryptovaluutat

Lohkoketju on tekniikka, jonka avulla järjestelmät voivat tuottaa ja ylläpitää yhteistä tietokantaa hajautetusti.[46] Lohkoketju koostuu listasta transaktioista, jotka jaetaan osallistujien kesken ja jonka eheys voidaan varmistaa lohkossa mukana kulkevilla kryptograafisilla tiiviisteillä. Näin osallistujat voivat luottaa toisiinsa tuntematta toisiaan etukäteen.

Lohkoketjuun voidaan lisätä uusia transaktioita vain listan päähän. Tällöin listaan kirjoitetaan listan nykyisestä päästä laskettu tiiviste, aikaleima ja transaktion tiedot. Uusi transaktio muodostaa listan uuden pään. Listassa olevia transaktioita ei voida käytännössä muuttaa, koska se vaatisi tiivisteiden laskemisen uudestaan muutoskohdasta listan loppuun saakka.

Lohkoketjua hallinnoi tyypillisesti vertaisverkko ohjelmistoja, jotka käyttävät jotakin sovittua konsensusalgoritmia päättääkseen, mikä transaktio liitetään seuraavaksi listaan ja kuinka se validoidaan.

Hajautetuissa järjestelmissä lohkoketju on kopioitu jokaiseen erilliseen järjestelmään kokonaisuudessaan. Nämä järjestelmät viestivät toisilleen uusien transaktioiden lisäämiseksi ja päätyvät konsensusalgoritmin avulla valitsemaan saman transaktion seuraavaksi listaan lisättäväksi.

Ensimmäinen lohkoketjuratkaisu julkaistiin vuonna 2008 Bitcoinin julkistamisen yhteydessä. Bitcoinia on seurannut joukko erilaisia lohkoketju-

[46] fi.wikipedia.org/wiki/Lohkoketju

pohjaisia valuuttoja, joita kutsutaan tiivisteen laskennan myötä krypto-valuutoiksi.

Lohkoketjuja käytetään edellä mainittujen kryptovaluuttojen lisäksi älykkäissä sopimuksissa ja digitaalisissa hallintasopimuksissa (non-fungible token, NFT).[47]

### 8.2.1 Lohkoketjujen energiankulutus

Lohkoketju on luonteeltaan energiatehoton ratkaisu, johtuen sen hajautetusta luonteesta ja tiedon moninkertaisesta tallentamisesta. Sen energiankulutus syntyy seuraavien toimien pohjalta:[48]

- **Konsensusalgoritmin ajokerrat** – osa konsensusalgoritmeista on laskentaintensiivisiä ja niiden kuluttama energiamäärä saattaa kasvaa verkkoon liitettyjen laitteiden määrän mukaan. Esimerkiksi Bitcoinin käyttämä proof-of-work -pohjainen algoritmi pohjautuu nimensä mukaisesti työn määrään ja vie merkittävän määrän maailman sähköenergiasta.

- **Toisteisuus** – koska kaikki lohkoketjuun liittyvät järjestelmät sisältävät lohkoketjun kokonaisuudessaan, lohkoketjun vaatima säilytystila kasvaa jokaisen uuden transaktion myötä. Koska ketju on muuttamaton, sitä ei voida lyhentää ilman sen integriteetin eli eheyden tuhoutumista.

  Lisäksi jokainen ketjua käsittelevä järjestelmä tekee itsenäisesti samat laskennalliset operaatiot jokaisen transaktion yhteydessä.

- **Tiedonsiirto** – lohkoketjua käsittelevien järjestelmien täytyy kommunikoida jatkuvasti toistensa kanssa päästäkseen konseksukseen seuraavasta lisättävästä transaktiosta. Järjestelmien määrän

[47] fi.wikipedia.org/wiki/NFT

[48] An analysis of energy consumption and carbon footprints of cryptocurrencies and possible solutions, www.sciencedirect.com/science/article/pii/S2352864822001390

kasvu kasvattaa myös niiden välistä tiedonsiirtoa ja yleensä laskee tiedonsiirron tehokkuutta. Tehokkuuteen vaikuttaa myös järjestelmien muodostaman verkon topografia – tiiviimpi verkko on energiatehokkaampi. Mittakaavaa tehottomuuten antaa Bitcoinin tiedonsiirron yli 98% toisteisuusaste. Toisin sanoen, alle 2% siirretystä tiedosta on uutta.

Koska lohkoketjuilla ei tulla koskaan pääsemään samoihin tehokkuuslukuihin kuin keskitetyillä järjestelmillä, pitkälti ratkaisun hajautetun luonteen ja kryptografisen laskennan takia, on syytä miettiä lohkoketjun käyttöä tarkkaan.

Mikäli lohkoketju ei anna aidosti etua verrattuna keskitettyyn ratkaisuun, sen käyttöä ei voi energiankulutuksen takia suositella. Lisäksi tulee huomioida, että lohkoketjupohjaiset ratkaisut ovat hajautetun luonteensa takia monimutkaisempia suunnitella, toteuttaa ja ylläpitää. Toisaalta ha- jautus antaa ratkaisulle paremman kokonaissaatavuuden kuin keskitetty järjestelmä, jonka toiminta on täysin riippuvainen keskeisen komponentin saatavuudesta.

### 8.2.2 Kryptovaluuttojen energiankulutus

Vaikka kryptovaluutat eivät olekaan yleensä osana sovelluksia, niiden käyttö aiheuttaa merkittävää energiankulutusta ja siksi ne on syytä käsitellä lohkoketjujen suurimpana käyttökohteena.

Kryptovaluuttojen energiankulutus pohjautuu edellisessä kappaleessa esiteltyyn lohkoketjun kulutusmalliin. Eri valuuttojen teknisessä toteutuksessa on kuitenkin otettu käyttöön hyvinkin erilaisia ratkaisuja, joilla on merkittäviä vaikutuksia käytännön energiankulutukseen.

Kryptovaluuttoja on tuhansia erilaisia[49], mutta niiden markkina-arvo putoaa nopeasti kärjen jälkeen. Elokuussa 2023 markkina-arvoltaan suurimmat valuutat olivat:[50]

1. Bitcoin 578 miljardia Yhdysvaltain dollaria
2. Ethereum 223 miljardia
3. Tether 83 miljardia
4. BNB 38 miljardia
5. XRP 34 miljardia

Listan kymmenennen valuutan markkina-arvo oli vain 7 miljardia dollaria. Kryptovaluutan energiankulutukseen vaikuttaa valitut teknologiat ja valuutan arvo – mitä suurempi arvo, sitä enemmän ihmiset haluavat osallistua kryptojen louhintaan.[51]

Konsensusalgoritmeista tehottomin on Bitcoinin käyttämä proof-of-work. Siinä järjestelmät laskevat tietyn määrän erilaisia hankalia laskutoimituksia ja saavat tällä työllä todistettua osallisuutensa seuraavan transaktion kytkentään. Tällä työllä ei ole muuta arvoa eikä sen tuloksia käytetä mihinkään. Kun Bitcoinia käsittelevien koneiden määrä ja sitä myöten laskentateho kasvaa, työtä lisätään vastaavassa suhteessa. Tällä pyritään varmistamaan, että kukaan – edes valtiollinen toimija – ei pysty tarjoamaan 51% verkon laskentatehosta ja siten kaappaamaan konsensusta.

Tämä näkyy suoraan Bitcoinin vaatimassa energiassa. Sen arvioidaan olevan vuositasolla elokuun 2023 lukujen pohjalta noin 107 TWh, mikä vastaa Alankomaiden energiankulutusta. Yksittäinen transaktio kuluttaa energiaa noin 635 kWh. Verrattuna luottokorttimaksuun, Bitcoinin trans-

[49] originstamp.com/blog/how-many-cryptocurrencies-are-there

[50] www.bankrate.com/investing/types-of-cryptocurrency/

[51] Bitcoin boom: What rising prices mean for the network's energy consumption, www.cell.com/joule/fulltext/S2542-4351(21)00083-0

aktio on noin 430 tuhatta kertaa tehottomampi.[52] Lisäksi on syytä huomioida Bitcoinin laskentaa tekevien laitteistojen tuottamisen jalanjälki. Laitteistot ovat pitkälle erikoistuneita laskentakoneita, joilla ei ole välttämättä järkevää uusiokäyttöä.

Toiseksi suurin kryptovaluutta Ethereum siirtyi syyskuussa 2022 proof-of-work -mallista proof-of-stake -malliin, jossa osallisuus arvioidaan siitä valuuttamäärästä, jota yksittäinen järjestelmä on valmis laittamaan eräänlaiseksi pantiksi ("stake") transaktion turvaamiseksi. Muutos oli suorastaan käänteentekevä. Proof-of-work -mallin arvioitu vuosikulutus oli juuri ennen proof-of-stake muutosta noin 83 TWh ja sen jälkeen vain 20 MWh. Yksit- täinen Ethereum-transaktio kuluttaa arviolta 0,03 kWh, joka vastaa noin 40 luottokorttimaksua.[53]

Valitettavasti muiden suurten kryptovaluuttojen energiankulutuksesta ei ole vastaavaa tietoa. Neljänneksi suurin BNB käyttää tehokasta proof-of-stake-authority -mallia ja lisäksi verkko on validoinnin osalta suljettu – käytössä on 21 palvelinta, jotka vastaavat uusien transaktioiden tarkastamisesta.[54] Toukokuussa 2022 yksittäisen transaktion energiankulutukseksi laskettiin 0,008 Wh, joka on vähemmän kuin luottokorttimaksun kulutus.[55]

Kuten yllä olevista esimerkeistä voidaan nähdä, kryptovaluuttojen energiankulutus vaihtelee merkittävästi valuutasta toiseen. Valitettavasti energiatehottomuus ei näy Bitcoinin suosiossa, vaan hukkaamme merkittävän määrän energiaa – sekä fossiilista että puhdasta – pitkälti spekulointikäytössä olevaan valuuttaan.

---

52 digiconomist.net/bitcoin-energy-consumption

53 digiconomist.net/ethereum-energy-consumption

54 www.adan.eu/en/publication/blockchain-protocols-and-their-energy-footprint/

55 opentaps.org/2022/03/24/estimating-the-energy-impact-of-the-binance-smart-chain/

Mikäli sinulla on Bitcoineja, ilmaston kannalta järkevin teko on jättää ne pysyvästi kryptolompakkoon. Koska jos myyt ne, syntyy uusi energiaa kuluttava transaktio ja luultavasti ostaja laittaa valuutan uudestaan kiertoon. Toinen kysymys on toki, että onko ihmisillä aidosti varaa tukea ilmastotyötä Bitcoin-omistuksillaan, jotka voivat olla merkittäviäkin.

Muutenkin kehotan miettimään kryptosijoittamisen mielekkyyttä. Normaaleja käyttötapauksia, esimerkiksi kryptovaluuttojen käyttöä maksuvälineenä, ei juurikaan ole, vaan kyse on lähinnä spekuloinnista. Jos taas haluat olla mukana kryptoissa, niin tutustu valuuttaan myös energiankulutuksen kautta.

## 8.3 Esineiden Internet (IoT)

Esineiden internet (IoT) tarkoittaa järjestelmiä, joilla laitteita voidaan etäseurata ja -ohjata niiden automaattisesti tuottaman tiedon pohjalta Internet- yhteyden avulla.[56] Puhutaan myös teollisesta internetistä ja älykodeista, jotka ovat esineiden internetin sovelluksia.

Esineiden internetin tuottama tieto pyritään jalostamaan järkevästi hyödynnettävään muotoon, esimerkiksi reaaliaikaisen analytiikan avulla valtava tietomassa muutetaan käyttökelpoisiksi tunnusluvuiksi ja niistä johdetuiksi tilannekatsauksiksi.

Internetiin kytkettyjen IoT-laitteiden määräksi arvioitiin vuonna 14,4 miljardia ja niiden määrän ennustetaan kasvavan 29,7 miljardiin kappaleeseen vuoden 2027 loppuun mennessä. Keskimääräinen arvioitu vuosikasvu on 16% vuodessa.[57] Tällä hetkellä machine-to-machine -yhteyksien arvioidaan olevan noin puolet Internetiin kytkettyjen laitteiden yhteyk-

[56] fi.wikipedia.org/wiki/Esineiden_internet

[57] State of IoT – Spring 2023, iot-analytics.com/product/state-of-iot-spring-2023/

sistä.[58] Yhteyksien määrissä älykotilaitteet ovat suurin segmentti ja Internetiin kytkeytyvät autot puolestaan nopeimmin kasvava segmentti.

Tämän päälle tulee laskea vielä suljetuissa verkoissa olevat IoT-laitteet, esimerkiksi tehdasverkossa toimivat sensorit. Kaikkinensa laitteiden määrä on kasvanut niin suureksi, että energiankulutuksen kannalta myös pienet liikahdukset ovat merkittäviä.

### 8.3.1 Energiankulutus

Tyypilliset IoT-laitteet, erilaiset sensorit, on suunniteltu energiatehokkaiksi. Ne voivat olla esimerkiksi paristo- tai akkukäyttöisiä. Anturi voi olla esimerkiksi upotettu betonivaluun tai koteloitu seinän sisään tarkkailemaan kosteutta. Tällöin on merkitystä, että pitääkö laitteen paristo vaihtaa vuoden vai kymmenen vuoden välein.

Jos anturi taas saa sähkön mitattavasta laitteesta, suunnittelussa energiankulutus ei olekaan enää niin merkittävä tekijä. Laitteista saatavan tiedon arvo on niin suuri, että energiankulutus ei ole käytännössä merkittävä tekijä, ellei se aiheuta yllä kuvattua manuaalista työtä.

IoT-antureiden kuluttamaa energian määrää voidaan säätää seuraavilla parametreilla:

- **Näytteenoton taajuus** – kuinka useasti laite tekee mittauksia. Mitataanko esimerkiksi hetkellistä tilannetta vai vaikkapa virtausta ajan yli. Tai tarkkaileeko laite vain raja-arvon ylittäviä tapahtumia.
- **Tiedonsiirron väylä** – laitteet voivat kytkeytyä Internetiin usealla eri tavalla, tyypillisiä ovat esimerkiksi WLAN- tai mobiiliverkkojen käyttö. Lisäksi kotiautomaatiossa käytetään muun muassa Zigbeetä ja Z-wavea. Kuten aiemmin on ollut puhetta, verkkojen energiankulutuksessa on merkittäviä eroja. Esimerkiksi mainitut kotiautomaation väylät ovat merkittävästi tehokkaampia kuin WLAN.

[58] Cisco Annual Internet Report (2018–2023) White Paper, www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

- **Tiedonsiirron taajuus** – siirretäänkö tietoa vastaanottajalle jokaisen näytteenoton yhteydessä vai koostetaanko isompi määrä näytteitä yhdeksi paketiksi, joka siirretään. Tällöin säästetään datan siirtämiseen liittyvää energiaa – esimerkiksi viestintäkanavan avaamiseen ja sulkemiseen liittyvät toimet – mutta vastaavasti menetetään reaaliaikaisuutta. Laite voi myös monimutkaistua ja näytteiden tallentaminen voi vaatia lisää tallennustilaa, mitkä kasvattavat laitteen valmistamisesta syntyviä päästöjä.
- **Näytteen koko** – kuinka paljon tietoa sisältyy näytteeseen. Tällä on pitkälti vaikutusta tiedonsiirron kulutukseen laitteen ja datan vastaanottavan järjestelmän välillä. Voi olla esimerkiksi järkevää pakata näyte ennen siirtoa, mikäli tiedonsiirto kuluttaa paljon energiaa verrattuna prosessointiin.

Nämä parametrit ovat tiukasti sidottuja mitattavan prosessin tai suureen luonteeseen ja käyttötapaukseen. On ensisijaisen tärkeää valita oikeanlainen järjestelmä laitteineen ja antureineen – vasta sen jälkeen tulee ryhtyä miettimään energiankulutuksen hillintää.

Antureista saatava data pitää tallentaa, sitä tulee tulkita ja analysoida, mahdollisesti suodattaa ja lopulta tästä syntyvästä informaatiosta voidaan vetää johtopäätöksiä – joko koneellisesti tai ihmisten kesken. Tämä prosessi kuluttaa myös energiaa ja kulutus kasvaa suhteessa käsiteltävän datan määrään.

On hyvä miettiä datalle säilytysajat ja mahdollisuuksien mukaan käyttää erilaisia datan määrää vähentäviä ratkaisuja, kuten esimerkiksi kohti menneisyyttä harvenia aikasarjatietokantoja. Niiden avulla voidaan tehdä analyyseja nykyhetkestä ja samalla säilyttää kuitenkin historialliset keskiarvot esimerkiksi poikkeamien havaitsemiseksi.

Kannattaa myös muistaa, että käyttämätön data on hukkaa. Jos liiketoiminta ei vaadi jonkun asian seuraamista IoT-tasolla, onkin järkevää miettiä toisenlaista lähestymistä asian ratkaisemiseen kuin varmuuden vuoksi säilöä anturidataa vuodesta toiseen.

## 8.4 Data

Kaikki edellä mainitut ratkaisut tuottavat ja käyttävät valtavia määriä dataa. Niiden lisäksi dataa syntyy esimerkiksi analytiikasta, sovellusten toiminnan instrumentoinnista ja ihmisten toiminnan seuraamisesta. Lisäksi ihmiset tuottavat jatkuvasti dataa valokuvina, videoina ja teksteinä vaikkapa erilaisten sosiaalisen median palveluiden avulla. Vastaavasti tämän datan tuottaminen, käyttäminen ja siirtäminen synnyttää lisää analytiikkadataa.

Dataa kerätään käytännössä kaikesta sähköisestä ympäriltämme. Puhutaan datavirroista, jotka laskevat datajärviin. Näistä sitten dataa pumpataan käsittelyyn, jotta saadaan liiketoimintaan liittyvää informaatiota. Tämä järjestely kuulostaa melkein runolliselta ja oikein hyödynnettynä se tarjoaakin paljon kilpailuetua dataa käsittelevälle yritykselle.

Mutta jos yritys taas ei osaakaan käsitellä dataansa oikein tai kerää aivan väärää dataa, synnytetään hukkaa. Tiedon keräystaajuus voi olla liian tiheä, sitä kerätään ja tallennetaan tehottomassa muodossa tai tietoa pidetään varmuuden vuoksi tallessa liian kauan.

Lisäksi yritysten tuottamat sovellukset tuottavat, siirtävät ja kuluttavat vuodesta toiseen enemmän dataa, toisin sanoen sovellusten data-intensiteetti kasvaa. Laitteiden sensorien tarkkuus kasvaa, jolloin ne tuottavat kerralla aina enemmän dataa – esimerkiksi kännykkäkuvien tiedostokoko kasvaa kameran megapikselien kasvun mukaan.

Myös sovellusten käyttäjien tottumukset muuttuvat ja maailman kommunikaatiosta yhä suurempi osa perustuu videoihin, jotka ovat informaatiotiheydeltään merkittävästi huonompia kuin kuvat tai teksti. Tässä informaatiotiheys tarkoittaa ihmiselle välittyvän informaation määrää suhteessa siirretyn tiedon tavumäärään. Samoin jokainen käyttäjä haluaa käyttää dataa haluamaansa tahtiin, jolloin sen broadcast-lähetys ei toimi, vaan jokaiselle käyttäjälle data lähetetään yksitellen unicastina. Tästä syystä esimerkiksi elokuvien ja televisio-ohjelmien striimaus verkosta on paljon tehottomampaa kuin niiden lähettäminen antenniverkon kautta.

Kaikki tämä johtaa datan määrän ja siirron kasvuun. Ja koska data luonteeltaan kumuloituu, eli sitä tallennetaan paljon mieluummin kuin poistetaan, kasvu on luonteeltaan jatkuvaa ja valitettavan nopeaa.

### 8.4.1 Globaali mittakaava

Globaalit datamäärät ovat valtavia ja kasvu huimaa vuodesta toiseen. IDC:n arvion mukaan vuonna 2018 dataa oli 33 tsettatavua ja määrä kasvaa vuoteen 2025 mennessä 175 tsettatavuun.[59] Tsettatavu (Zt) on $10^{21}$ tavua eli miljardi teratavua.[60]

Samassa IDC:n raportissa arvioidaan datan tallennuksen siirtyvän merkittävässä määrässä päätelaitteista keskitettyihin datavarastoihin, eli datakeskuksiin ja pilveen. Datan luonti kuitenkin tapahtuu edelleen pääasiassa päätelaitteissa, vaikka keskitetysti ja reunalla – eli verkossa tai hajautetuissa palvelimissa – tapahtuvan datan luonti kasvaakin hieman. Tämä johtaa puolestaan kasvaviin tiedonsiirtotarpeisiin päätelaitteiden ja keskitettyjen ratkaisujen välille.

Kuluttajien tuottama datamäärä oli raportin mukaan vuonna 2017 47% kaikesta datasta ja sen arvioidaan vähenevän tasaisesti ja olevan vain 36% vuonna 2025. Loput datasta on yritysten tuottamaa dataa.

Datan tallentamisen energiankulutuksesta ei valitettavasti löydy tilastotietoa ja erilaiset arviot ovat vanhoja sekä sisältävät tallentamisen lisäksi myös tiedon siirtämisen kustannukset. Tiedonsiirrosta on puhuttu aiemmin tässä kirjassa ja alla käsitellään siirtomääriä.

Tallennuksen arvioinnissa joudutaan käyttämään laskentakaavoja, jotka pohjautuvat tallennusmedian – eli joko kova- tai SSD-levyjen – keski-

[59] The Digitization of the World From Edge to Core, www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[60] fi.wikipedia.org/wiki/Tavu_(tietotekniikka)

arvoiseen kapasiteettiin ja laitteiden kuluttamaan energiaan.[61] Vuoden 2020 arvoilla kovalevyjen kulutus on 0,65 wattituntia per teratavutunti (teratavun säilyttäminen tunnin ajan) ja SSD-levyjen kulutus 1,2 Wh / Tth. Tästä voidaan laskea vuosittainen kulutus 5,7 kWh / Tta ja 10,5 kWh / Tta.

IDC:n arvio tämän luvun kirjoitusvuoden – 2023 – datan globaaliksi kokonaismääräksi on 100 Zt, jolloin sen tallentaminen kertaalleen vie laskelman mukaan 570 TWh energiaa vuodessa.

Globaali vuotuinen sähkönkulutus on 25 530 TWh.[62] Näin globaalin datan sähkönkulutus olisi jo 2,2% maailman kulutuksesta. Ja data olisi tallennettu vain kertaalleen ja täysin optimoidusti, sitä ei siirrettäisi tai käsiteltäisi lainkaan. Voidaankin siis todeta, että laskentakaava ei voi pitää paikkaansa, vaan ampuu merkittävästi yli.

Lisäksi laskelmassa tulisi huomioida datan replikointi. Erityisesti pilvipalveluissa ja datakeskuksissa datasta säilytetään varmuuden vuoksi useampaa kopiota, replikaa, joilla varmistetaan datan saatavuus myös mahdollisen laiterikon yhteydessä. Pilvipalveluiden käyttämien replikoiden määristä ei ole tarkkaa tietoa, mutta erään laskelman mukaan ne vaihtelevat yhden ja kuuden välillä.[63]

Globaalin kuukausittaisen tiedonsiirron arvioitiin UNCTAD:n raportin mukaan olleen vuonna 2019 180 eksatavua ja kasvaneen vuonna 2020 230 eksatavuun.[64] Vuonna 2026 tiedonsiirron arvioidaan olevan noin kolmin-

[61] www.cloudcarbonfootprint.org/docs/methodology/

[62] World Energy & Climate Statistics – Yearbook 2023, yearbook.enerdata.net/electricity/electricity-domestic-consumption-data.html

[63] Cloud Carbon Footprint Replication Factors, docs.google.com/spreadsheets/d/1D7mIGKkdO1djPoMVmlXRmzA7_4tTiGZLYdVbfe85xQM/edit#gid=735227650

[64] UNCTAD Digital Economy Report 2021: Cross-border data flows and development: For whom the data flow, unctad.org/system/files/official-document/der2021_en.pdf

kertainen eli 780 Et. Eksatavu on $10^{18}$ tavua, eli miljoona teratavua. Vuotuinen datasiirto vuonna 2019 on 2.2 Zt, joka tuntuu kovin pieneltä IDC:n 40 Zt arvioon saman vuoden kokonaisdatamäärästä.

Samassa raportissa arvioidaan mobiilitiedonsiirron kasvavan kännyköiden ja IoT-laitteiden määrän kasvun myötä nopeammin kuin kiinteän verkon tiedonsiirron. Vuonna 2026 mobiilitiedonsiirto olisi noin kolmannes kaikesta tiedonsiirrosta.

Kuluttajien tiedonsiirto jatkaa kasvuaan sekä laitteiden käyttöajan että datan käytön intensiteetin kasvamisen pohjalta. Arthur D. Littlen Eurooppaan keskittyvän raportin mukaan verkossa vietetty aika näyttää tasoittuvan kehittyneissä maissa noin kolmeen–neljään tuntiin päivässä, mutta käytön intensiteetti jatkaa kasvuaan useasta eri syystä:[65]

- Videoiden resoluutioiden paraneminen
- Videoiden laajempi käyttö sosiaalisessa mediassa
- Internetin käytön laajeneminen uusiin käyttötapauksiin – kuten ihmisten tapaaminen, konserteissa käynti tai ostokset
- AR- ja VR-ratkaisujen käyttö
- Tekoälyn tuottaman sisällön kasvu

Kaikista suurin energiakulutuksen kasvun ajuri on videoiden katselu. Samaisen raportin mukaan videoiden katselu muodosti vuonna 2022 60% mobiilista tiedonsiirrosta ja osuuden arvioidaan kasvavan vuoteen 2030 mennessä 72 prosenttiin. Samassa ajassa yksittäisen käyttäjän kokonaistiedonsiirto mobiilissa arvioidaan kasvavan 15 gigatavusta kuukaudessa 75 gigatavuun, vuosittaisen kasvun ollessa 25%. Eli vuonna 2022 videot kuluttivat keskimäärin yhdeksän gigatavua mobiilikaistaa kuukaudessa ja vuonna 2030 videoiden datamäärän arvioidaan olevan 54 gigatavua.

Vastaavasti kiinteillä yhteyksillä – per kotitalous – siirrettiin keskimäärin 225 gigatavua kuukaudessa ja sen arvioidaan kasvavan 900 gigatavuun

[65] The Evolution of Data Growth in Europe, www.adlittle.com/en/insights/report/evolution-data-growth-europe

vuoden 2030 aikana – vuosittaista kasvua 20%. Videoiden osuus on vastaavasti 67% ja 74%, eli videoita siirretään 151 Gt vuonna 2022 ja 666 Gt vuonna 2030.

Vaikka tiedonsiirron energiatehokkuus tulee varmasti uusien teknologioiden käyttöönoton myötä paranemaan, sillä ei kuitenkaan näillä näkymin pystytä yksin vastaamaan siirretyn datan määrän kasvuun. Toivottavasti tulevaisuudessa pystymme vähentämään merkittävästi verkkojen käyttämän energian hiilidioksidipäästöjä siirtymällä uusiutuviin energialähteisiin.

Kuluttajien käyttäytymisen ja datankäytön muutokset eivät ole ainoita ajureita datamäärän kasvussa. Monet nosteessa olevat ratkaisut, kuten esimerkiksi tekoäly tai verkkomainonta, vaativat merkittäviä määriä dataa toimiakseen.

Tekoälyn opetusdatat ovat valtavia ja tekoälyjen tarkkuusvaatimusten kasvaessa datamäärät jatkavat kasvuaan. Samoin tekoälyn opetusmateriaalia täytyy jatkuvasti kerryttää, sillä muuten tekoäly jäisi auttamatta jälkeen maailman tapahtumista ja se ei osaisi tuottaa viimeisten taidevirtausten mukaisia kuvia tai videoita.

Esimerkiksi Common Crawlin tuottaman Internet-arkiston ja -datakokoelman koko oli lokakuussa 2022 380 teratavua.[66] Tämä arkisto muodostaa noin 60% ChatGPT:n taustalla olevan GPT-3 -tekoälymallin opetusdatasta.

Lukujen osittaisesta ristiriitaisuudesta huolimatta kaikki luvut ovat merkittävän suuria ja jokainen niistä on voimakkaassa kasvussa vuodesta toiseen. Datamäärien hillintä onkin kestävän IT:n suurimpia haasteita, koska datan siirto, tallennus replikointeineen ja käsittely synnyttävät suuren osan IT-alan energiankulutuksesta ja siten myös hiilidioksidipäästöistä.

[66] en.wikipedia.org/wiki/Common_Crawl

## Tiivistettynä

1. Perussovelluskoodin lisäksi merkittävästi energiaa kuluttavat ratkaisut, kuten tekoäly (AI) ja lohkoketjut, on huomioitava erikseen niiden suuren energiankulutuksen vuoksi. Vaikka niiden käyttö tarjoaa hyötyä, on hyvä ymmärtää niiden energiavaikutus.

2. Tekoäly on saanut huomiota ChatGPT:n ja DALL-E:n julkaisun myötä, mutta nämä ratkaisut kuluttavat huomattavasti energiaa. Tekoälyn koulutusmateriaalin hankinta, opettaminen ja käyttö kaikki vaikuttavat energiankulutukseen ja hiilidioksidipäästöihin.

3. Lohkoketjun hajautettu luonne ja moninkertainen tietojen tallennus tekevät siitä energiatehottoman. Kryptovaluutat kuluttavat myös merkittävästi energiaa – tosin niiden toteutustavat vaikuttavat suuresti energiatehokkuuteen.

4. IoT-laitteet tuottavat dataa hajautetusti, ja niiden energiankulutus riippuu useista tekijöistä – kuten näytteenottotaajuudesta ja datan määrästä, siirtotavasta ja jatkokäsittelystä. Kasvava IoT-laitteiden määrä herättää huolta energian käytöstä.

5. Datan tuotanto, siirto ja kulutus kasvavat nopeasti. Maailmanlaajuiset datavolyymit ovat valtavia, ja datan replikointi lisää energiankulutusta. Tekijät kuten videoiden kulutus ja uudet ratkaisut kuten tekoäly vaikuttavat datan kasvuun ja energiankulutukseen merkittävästi.

# 9 Vaikutuksen arviointi

Edellisissä kahdessa luvussa mainitut optimointitavat ja energiaa merkittävästi kuluttavien komponenttien säätö saattavat auttaa vähän tai paljon, riippuen ohjelmiston luonteesta ja sisäisestä rakenteesta. Osa niistä voi olla hyvin helppoja ja toiset taas äärimmäisen vaikeita. Tässä luvussa käydään läpi pari erilaista tapaa arvioida muutoksia ja kustannuksia, jotta voidaan tehdä informoituja päätöksiä muutosprojektin käynnistämisestä.

Euromääräinen arviointi on yleensä tehokkainta, koska tarpeeksi isoilla ja uskottavilla luvuilla saa aina ylimmän johdon huomion. Sekä talous- että toimitusjohtajat jäsentävät maailmaa pitkälti lukujen kautta ja jos heidät saa mukaan idean taakse moni este on jo ylitetty. Valitettavasti euromääräisille muutoksille ei voida antaa mitään yleispätevää kaavaa aivan perusteita lukuunottamatta.

Lisäksi sovelluksia toteutetaan erilaisiin tarpeisiin. Harvojen aika ajoin käyttämän ratkaisun optimointiin ei voida panostaa samassa laajuudessa kuin massoille suunnattuun ja päivittäin käytössä olevaan sovellukseen. Sovelluksen päivittäisiä käyttäjiä tai ajoaikaa voidaan käyttää kertoimena vaikutuksen arvioinnissa.

Koska toteutuskielestä riippumatta ohjelmiston käyttämä aika korreloi vahvasti käytettyyn energiaan, voidaan ajoajan tehostamisen todeta pienentävän palvelun energian käyttöä tai pilvipalvelun kustannuksia vastaavassa suhteessa. Tästä säästöstä voidaan helposti johtaa euromääräinen summa. Huomaa, että säästö kumuloituu joka vuosi käytön pysyessä sa-

malla tasolla. Ja jos käyttö kasvaa, niin kumuloitunut säästö kasvaa vastaavasti vuosi vuodelta.

Osa säästöistä on myös kalliiden muutosten, kuten kahdentamisen tai hajauttamisen, siirtämistä myöhemmäksi. Kun palvelu pystyy palvelemaan enemmän yhtäaikaisia käyttäjiä, se voi pysyä pitempään yksinkertaisena kuin vähemmän käyttäjiä palvelemaan pystyvä ratkaisu.

## 9.1 Vaikutus vs. työmäärä

Mikäli muutoksen vaikutusten kvantifiointi on vaikeaa, asiaa voi lähestyä myös nelikentän, kuva 3 alla, avulla. Jokaisesta ehdotetusta muutoksesta arvioidaan siihen käytettävä aika asteikolla pieni–suuri ja vastaavasti muutoksen tuottama energiansäästö eli vaikutus samalla asteikolla.

Jokainen organisaatio voi itse päättää, mitä pieni tai suuri työmäärä tai vaikutus heille tarkoittaa. Yritysten tarpeet ja tilanteet ovat erilaisia, joten mitään yleispätevää mittaristoa ei voida antaa.

Jos yritys esimerkiksi toimii energiaintensiivisellä alalla, kuten metallien jalostuksessa tai betonin valmistuksessa, prosentin säästö pääprosessin energiankulutuksesta tietotekniikan avulla voi olla niin suuri säästö, että sovelluksen viemä energia on toissijaista. Vastaavasti kokonaan sovelluksen päälle perustavan yrityksen energiansäästö voi olla heille valtava, vaikka se olisikin pienempi kuin edellisen esimerkin valimon säästö.

Muutoksen vaikutuksen tulee olla aidosti vähenteinen, eli vastaavaa säästöä ei saada aikaiseksi ilman muutosta eikä muutos tapahdu joka tapauksessa – muuten organisaatio sahaa itseään linssiin tai harrastaa viherpesua. Myös useamman mahdollisen muutoksen yhteenlasketut vaikutukset eivät välttämättä ole yhtä suuret kuin jokaisen muutoksen yksittäiset vaikutukset. Tämä on hyvä ottaa huomioon, kun muutoksista päätetään.

Koska sovelluskehittäjätkin ovat vain ihmisiä, heillä saattaa olla erilaisia ajattelun pinttymiä, jotka joko suosivat tai hylkivät tietynlaisia muutoksia. Joskus tällainen vinoutuma on selkeää henkilölle itselleen, mutta moni on

myös tiedostamatonta. Tästä syystä on järkevää pyytää ainakin kaksi arviota eri asiantuntijoilta samasta muutoksesta. Jos ajatukset menevät pahasti ristiin, niin tästä syntyy varmasti mielenkiintoinen keskustelu, jossa kaikkien ajattelu täsmentyy.

Alla on esitetty yksinkertainen nelikenttä ja muutamia erilaisia päätöksiä työmäärän ja saavutettavan vaikutuksen pohjalta.


*Kuva 3. Työmäärän ja vaikutuksen arviointi nelikentässä.*

## 9.2 Vaikutus vs. käyttökokemus

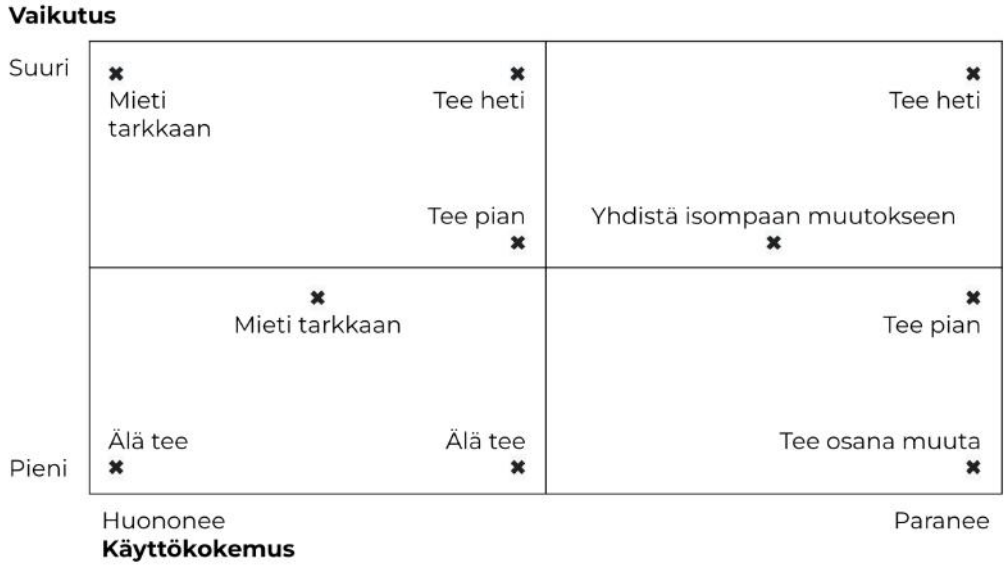
Pelkkä vaikutuksen arviointi sovelluskehittäjien työmäärää vastaan ei riitä, koska muutoksella voi olla suurempia vaikutuksia kuin pelkkä käytetty työmäärä, esimerkiksi käyttökokemuksen muutos. Mikäli näin ei ole ja muutos on puhtaan tekninen, seuraavaa nelikenttää ei tarvita. Jos taas käyttökokemus muuttuu, se on syytä arvioida vastaavasti suhteessa oletettuun vaikutukseen.

Nelikentässä käyttökokemus voi huonontua tai parantua. Helposti ajatellaan, että energiankulutuksen hillitsemiseksi tehdyt toimet huonontavat käyttökokemusta – kyse on säästöistä ja niillä on huono konnotaatio

ihmisten mielissä. Mutta jos muutos vähentääkin esimerkiksi käyttäjältä kysyttäviä tietoja tai muuten yksinkertaistaa palvelun käyttöä, muutos voi olla merkittävän positiivinen.

Tätä nelikenttää ei kannata jättää kehittäjien täyttämisen vastuulle, koska heillä ei välttämättä ole osaamista tai näkemystä arvioida käyttökokemuksen muutosta. Tällöin muutoksen arviointi tapahtuu helposti mututuntumalla tai se peilaa käyttöliittymän muuttamisen vaikeutta.

Toisekseen, arvioinnin oikeanlainen tekeminen tarjoaa myös uusia lähtöjä muutoksen viestinnälle käyttäjille. On varsin erilainen tilanne huomata videoiden laadun heikentyneen kuin lukea aiheesta viesti, jossa kerrotaan muutoksesta ja sen vaikutuksesta ympäristöön.




*Kuva 4. Käyttökokemuksen muutoksen ja vaikutuksen arviointi nelikentässä.*

## Tiivistettynä

1. Mittaa muutokset uskottavien ja mieluiten euromääräisten mittarien avulla. Näin saat helpommin johdon ja erityisesti talousjohtajien ja toimitusjohtajien huomion, koska he tekevät päätöksiä numeroiden kautta. Harkitse vuosittaisten säästöjen laskemista päätösten pohjaksi.

2. Arvioi muutoksia vaikutus vs. työmäärä -matriisin avulla, perustuen työmäärään ja energiansäästöön. Sovita matriisin määritelmät "pieni" ja "suuri" organisaation tarpeiden mukaisesti.

3. Arvioi muutoksia käyttäen käyttökokemuksen muutos vs. työmäärä -matriisia ottaen huomioon käyttökokemuksen parannukset tai heikennykset. Huomioi muutosten arvioinnissa myös viestinnän tarve ja käyttäjien oletetut reaktiot.

4. Ohjelmiston suoritusaikaa lyhentämällä vähennetään energiaa kulutusta ja pilvipalveluiden kustannuksia vastaavasti; säästöt voidaan helposti kvantifioida.

# 10 Hiilineutraalius

Hiilineutraalius on monen organisaation julkinen tavoite. Sillä on vaikutuksia organisaatioiden hankkijoihin, joilta ostettujen palveluiden hiilijalanjälki lasketaan osaksi ostajan jalanjälkeä. Koska IFRS-raportoinnissa tulee jatkossa olla hiilijalanjälki mukana kaikilta kolmelta laajuusalaltaan, kaikkien IFRS:ää noudattavien yritysten ja heidän alihankkijoidensa hiilijalanjälki tulee laskea viimeistään muutaman vuoden päästä.

Helpointa tietysti on, jos organisaatio on hiilineutraali ja jalanjälki on nolla. Tyypillisesti hiilineutraalius saavutetaan kolmessa vaiheessa:

1. **Lasketaan oman toiminnan hiilijalanjälki.** Se lasketaan yleisesti GHG-protokollan[67] mukaisesti ja se jakaantuu kolmeen laajuusalaan:

    a. Scope 1 – Yrityksen omien toimintojen takia paikan päällä syntyvät päästöt.

    b. Scope 2 – Epäsuorat ostoenergian liittyvät päästöt, esimerkiksi käytetyn sähkön- ja lämmöntuotannon päästöt.

    c. Scope 3 – Myytyjen tuotteiden käytöstä, tavaroiden ja palveluiden hankinnasta syntyneet päästöt, eli epäsuorat päästöt. Tämä luokka jakaantuu viiteentoista kategoriaan, joista

[67] Greenhouse Gas Protocol, ghgprotocol.org

päästölaskentaan valitaan yrityksen toiminnan kannalta keskeisimmät.

2. **Minimoidaan hiilijalanjälkeä.** Tämä on itse asiassa koko prosessin tärkein vaihe. Kun jalanjäljen eri komponentit ovat tiedossa, niihin voidaan kohdistaa erilaisia toimia. Esimerkiksi vähennetään polttoaineen kulutusta optimoimalla logistiikkaa, vähennetään työmatkoja siirtymällä videopuheluihin tai sallitaan jatkuvien etätöiden tekeminen. Eteneminen tulee aloittaa isoimmista päästölähteistä, joihin on mahdollista vaikuttaa. Esimerkiksi toimiston ilmastoinnin tai lämmityksen energialähteen vaihtaminen ei välttämättä ole mahdollista vuokratuissa toimistoissa, vaikka energiankulutus voi ollakin merkittävää.

   Minimointi on syytä tehdä useammassa vaiheessa, koska kaikkia muutoksia ei voida tehdä yhtä aikaa. Tyypillisesti laskenta tehdään vuosittain, joten minimoinnille sopiva aikaraami on samoin vuosi. Silloin nähdään edellisen vuoden toimien vaikutukset ja tietojen pohjalta voidaan suunnitella seuraavia minimointeja. Alan edelläkävijäyritykset ovat käyttäneet minimointiin jo yli vuosikymmenen ja heilläkin on vielä mahdollisuuksia minimoida omaa hiilijalanjälkeään.

3. **Kompensoidaan päästöt, joita ei voida minimoida**. Kompensointimenetelmiä on useita. Pääajatuksena on poistaa hiilidioksidia ilmakehästä kompensoitavan määrän verran. Tällöin lopputuloksena on laskennallisesti hiilineutraali toiminta.

   Kompensointia on syytetty anekaupaksi ja alalla toimii laadullisesti kirjavia toimijoita. Suomessa eri tahot ovat heränneet tähän ongelmaan ja tulossa onkin lisää ohjeistusta ja jossakin vaiheessa myös vaatimuksia kompensointipalvelujen valintaan ja toteukseen. Kompensoinnin etu on, että se asettaa päästöille hinnan, joka näkyy yrityksen tuloslaskelmassa.

Hiilijalanjälkilaskennan kannalta ohjelmistot kuuluvat kolmanteen laajuusalaan ja niiden laskenta mukaan hiilijalanjälkeen on ainakin vielä yrityksen

oma päätös. Osa teollisista toimijoista esimerkiksi laskee vain laajuusalat 1 ja 2 omaan jalanjälkeensä. ICT-teollisuudessa käytännössä kaikki päästöt ovat laajuusalassa 3, ellei toimija ole datakeskus tai teleoperaattori.

Sovellusten hiilijalanjälki voidaan jakaa kahteen osaan: sovelluksen toteutus ja ylläpito sekä sovelluksen käyttämän energian ja laitteiden hiilijalanjälki.

## 10.1 Toteuttamisen ja ylläpidon jalanjälki

Sovelluksen toteuttamisen hiilijalanjälki on pitkälti vastaava kuin normaalin toimistotyön jalanjälki. Ohjelmistokehittäjät käyttävät tietokoneita vastaavalla tavalla kuin muutkin toimistotyöläiset. Lisäksi tulee laskea sovelluskehitystä tukevien järjestelmien – versionhallinta, testipalvelimet, tiketöinti, julkaisuputki (“build pipeline”) ja niin edelleen – jalanjälki.

Projektit voivat jakaa resursseja, joiden jalanjälki tulisi jakaa käytön mukaisessa suhteessa. Tämä ei välttämättä ole helppoa, joten myös erilaiset arviot ovat täysin riittäviä. Olennaista on varmistaa, että jalanjälki jaetaan kokonaisuudessaan, mutta ei kuitenkaan moninkertaisesti.

Keskustelu sovelluskehittäjän jalanjäljestä versus sovelluksen jalanjäljestä on, kuten aiemminkin totesin, pitkälti merkityksetöntä. Molempia pitää pienentää ja pienennykset ovat hyvin harvoin kytköksissä toisiinsa. Esimerkiksi sovelluskehittäjän pyöräily töihin tuskin vähentää hänen kykyään tehdä vihreää koodia. Pikemminkin päinvastoin, kun lihakset tekevät enemmän työtä, aivot saavat lisää happea.

## 10.2 Sovelluksen jalanjälki

Sovelluksen jalanjälki koostuu käytetystä energiasta ja tarvittavien laitteiden elinkaaripäästöistä. Joskus energiankäyttö sisältyy elinkaareen, joten tässä kannattaa olla tarkkana, ettei tule tehtyä tuplalaskentaa.

Sovelluksen energiankulutus korreloi pitkälti sovelluksen käyttämän ajan kanssa, riippumatta toteutuskielestä. Eli nopeammin toimiva sovellus on

myös energiatehokkaampi. Samoin siirretyn tiedon määrä vaikuttaa jalanjälkeen siirtotien mukaisesti. Tämä kirja on pitkälti kirjoitettu sovelluksen tehokkuuden parantamiseksi ja jotta vältetään loputon rekursio, tässä luvussa ei käsitellä aiempia aiheita uudestaan.

## 10.3 Koodia Suomesta hiilineutraaliusmerkki

Koodia Suomesta ry.[68] on lanseerannut vuoden 2022 alussa hiilineutraaliusmerkin[69], jolla ohjelmistoyritykset voivat kertoa oman hiilijalanjälkensä nollaamisesta. Tulee huomata, että merkki kattaa toteuttamisen ja ylläpidon hiilijalanjäljen. Sovelluksen oma jalanjälki huomioidaan, mikäli se kuuluu yrityksen toimintaan – esimerkiksi SaaS-palvelutaloilla tarjottu sovellus on heidän jalanjälkensä sisällä. Vastaavasti ohjelmistokonsultointitalojen tuottamat sovellukset ovat heidän asiakkaidensa jalanjäljessä.

Merkin tarkoituksena on auttaa myös ostajia tekemään ilmastoviisaampia hankintapäätöksiä. Koodia Suomesta toivoo merkin olevan jonakin päivänä tarpeeton, koska kaikki ohjelmistokehitys on silloin hiilineutraalia.

## Tiivistettynä

1. Hiilineutraalius on tavoite monelle orhganisaatiolle ja sen merkitys vain kasvaa tulevaisuudessa. Neutraaliustavoite vaikuttaa hankintoihin ja raportointiin, ja siihen kuuluu kolme osa-aluetta: päästöt toiminnasta ja ostetusta energiasta sekä epäsuorat päästöt.

[68] koodiasuomesta.fi/

[69] koodiasuomesta.fi/yhdistys/hiilineutraaliusmerkki/

**2** Hiilineutraalius saavutetaan laskennan, vähentämisen ja päästöjen kompensoinnin avulla. Vähentäminen kohdistuu keskeisiin päästölähteisiin ja se tehdään vuosittain. Kompensaatiomenetelmät vaihtelevat laadultaan ja niiden valinnassa kannattaa olla tarkkana.

**3** Ohjelmiston hiilijalanjälki kuuluu laajuusalaan kolme. Jalanjälki kattaa sekä toteutuksen ja ylläpidon että energian ja laitteiston käytön.

**4** Koodia Suomesta on julkaissut hiilineutraaliusmerkin ohjelmistoyrityksille. Merkki kattaa ohjelmistojen toteutuksen ja ylläpidon hiilijalanjäljen ja pyrkii tukemaan ilmastoystävällisten ohjelmistohankintojen tekemistä.

# 11 Suositukset

Tässä luvussa esitellään lyhyet suositukset tärkeimmille vihreän sovelluskehityksen osapuolille. Jokaista suositusta ei ole tarkoitus noudattaa pilkulleen, vaan asiaa tulee arvioida oman tilanteen kautta. Yksiselitteistä ratkaisua ei ole. Lähimmäksi tällaista pääsee proof of work -kryptovaluuttojen käytön lopettaminen, koska ne syövät tolkuttomasti energiaa suhteessa hyötyyn.

## 11.1 Sovelluskehittäjille

Tuotettu lähdekoodi ja sitä myöten valmis ohjelmisto on niin energiatehokas kuin sen kehittäjät osaavat ja haluavat tehokasta koodia kirjoittaa. Järjestelmän vaatimukset ja rajaukset luovat tietysti energiankulutukselle raamit, joiden sisällä koodin energiatehokkuus voi liikkua. Ylärajaa tehottomuudelle ei periaatteessa ole olemassa, joten raamien sisälläkin on oleellista kehittää energiatehokkaita ratkaisuja.

Kehittäjiä voidaan motivoida energiatehokkuuteen erilaisilla ulkoisilla mekanismeilla, mutta sisäinen motivaatio on olennaista herättää pysyvien muutosten aikaansaamiseksi.

Toimi seuraavasti:

- Ota asia vakavasti, ryhdy toimeen samoin tein.
- Selvitä miten kehittämäsi sovellus kuluttaa energiaa ja tunnista parhaat kohdat tehdä energiaa säästäviä muutoksia.

- Minimoi tiedonsiirto- ja tallennustarpeet.
- Pohdi käyttämiesi kirjastojen järkevää käyttöä. Refaktoroi turha koodi mahdollisuuksien mukaan ulos. Älä rakenna turhan monimutkaisia rakennelmia, jos yksinkertaisempikin ajaa saman asian.
- Mieti muutostesi vaikutuksia ja suhteuta ne arvioituun työmäärään ja mahdollisiin muutoksiin käyttäjäkokemuksessa. Ota tarvittaessa muita asiantuntijoita, kuten tuoteomistajia ja UX-suunnittelijoita mukaan keskusteluun.
- Haasta liiketoimintaa ja suunnittelijoita heidän ehdottaessaan uusia toiminnallisuuksia. Keskustele aktiivisesti suunnittelijoiden kanssa heidän suunnitelmiensa energiankulutuksesta. Käy avointa dialogia tavoitteista, jotka huonontavat energiatehokkuutta.
- Mikäli toteutus sisältää tekoäly- tai lohkoketjukomponentteja, tutustu niiden energiankulutukseen. Tee tarvittaessa omia mittauksia, jos tietoa ei ole saatavilla. Jos ostat nämä palveluna, pyydä toimittajalta energiankulutustietoa ja ohjeistusta, kuinka pienentää ratkaisun energian käyttöä.
- Jos olet senioritason tekijä, pidä huolta vähemmän kokeneista kehittäjistä ja auta heitä ymmärtämään ongelmia ja niihin sopivia ratkaisuja.
- Ymmärrä, että kaikkea ei voi saada ja joskus tehotonta koodia on pakko tuottaa tai jättää paikoilleen. Kaikille muutoksille ei löydy maksajaa. Tästä ei kannata hermostua, vaan keskittää ponnistelut seuraaviin muutoksiin.
- Älä sotke omaa hiilijalanjälkeäsi tuottamasi sovelluksen hiilijalanjälkeen. Ne ovat kaksi eri asiaa ja niihin on erilliset, riippumattomat ratkaisut. Ota kaikki mahdollinen käyttöön.
- Pidennä käyttämiesi laitteiden uusimissykliä. Vältä vaihtamasta täysin toimivaa laitetta uuteen. Korjauta laitteita mahdollisuuksien mukaan.

- Osallistu teollisuudenalaa koskevien ratkaisujen kehittämiseen ja keskusteluun ohjelmistojen energiatehokkuudesta. Jaa kokemuksiasi ja ideoitasi.

### 11.1.1 Komponenttien kehittäjille

Mikäli sovelluskehittäjä toteuttaa muihin sovelluksiin upotettavaa komponenttia tai kirjastoa, edellä olevat suositukset pätevät suoraan. Sen lisäksi on hyvä miettiä komponentin energiatehokkuutta – erityisesti, jos komponentti on suosittu.

Esimerkiksi SQLite[70] on asennettu kaikkiin Android- ja iOS-laitteisiin, Applen Mac-tietokoneisiin, Windows 10 -järjestelmiin ja kaikkiin laajassa käytössä oleviin selaimiin. Arvio tietokantojen määristä lasketaan biljoonissa. Joten pienetkin energiankulutusten muutokset ovat merkittäviä, koska ne kertautuvat valtavilla kertoimilla.

Toki kaikki komponentit eivät ole näin suosittuja, mutta se ei silti estä energiatehokkuuden miettimistä projektissa.

Toimi seuraavasti:

- Käy aktiivista keskustelua tehokkuudesta projektissa.
- Varmista koherentti ja projektin läpi kulkeva tapa käsitellä tehokkuutta ja energiankulutusta.
- Rakenna mittausjärjestelmä ja suorita tehokkuustestit muutoksesta toiseen. Mikäli muutokset ovat merkittäviä tai tehokkuus laskee trendinomaisesti, puutu asiaan.
- Puutu komponentin käyttämien kirjastojen energiatehokkuuden muutoksiin aktiivisesti.
- Julkaise mittaustulokset. Jaa kokemuksiasi ja ideoitasi.

[70] www.sqlite.org/mostdeployed.html

## 11.2 Suunnittelijoille

Sovellus ei ole pelkästään koodia, vaan sen käyttäjälle suunnatut toimintojen ja niiden visuaalisen ulkoasun suunnittelussa määritetään myös sovelluksen tehokkuutta, joskus jopa enemmän kuin lähdekoodissa.

Suunnittelijoiden tulee tuntea vastaavalla tavalla vastuunsa sovelluksen energiatehokkuudesta kuin kehittäjien ja arkkitehtien.

Toimi seuraavasti:

- Ota asia vakavasti ja ryhdy toimeen.
- Ymmärrä sekä liiketoiminnan tarpeet että käyttäjien toiveet. Pyri ratkaisemaan nämä mahdollisesti ristiriitaiset tarpeet energiatehokkaasti.
- Minimoi käyttäjän mahdollisuudet virheisiin. Suunnittele ja kirjoita käyttöliittymä mahdollisimman selkeäksi ja suoraviivaiseksi. Pidä huolta saavutettavuudesta, sillä se vähentää rajoitetun liikkuvuuden tai muiden haasteiden kanssa toimivien käyttäjien virheitä.
- Mieti millainen ratkaisu olisi mahdollisimman minimaalinen. Haasta liiketoimintaa ja sovelluskehittäjiä heidän ehdottaessaan uusia toiminnallisuuksia. Käy avointa dialogia tavoitteista, jotka huonontavat energiatehokkuutta.
- Keskustele aktiivisesti sovelluskehittäjien kanssa suunnitelmiesi energiankulutuksesta. Kasvata omaa ymmärrystäsi sovelluksen toiminnasta ja hahmota, mikä on kallista ja mikä halpaa energiankulutuksen kannalta.
- Mieti, voidaanko toiminnallisuuksia korvata joko osittain tai kokonaan ohjeteksteillä ja -kuvilla, jotka ohjaavat käyttäjää. Suunnittele sisällöt mahdollisimman energiatehokkaaksi, pitäen kuitenkin huolta sisällön informatiivisuudesta ja arvosta käyttäjälle.

- Yritä poistaa uutta koodia vaativia ongelmia kokonaan sen sijaan, että etsisit niihin ratkaisua. Kaikkia ongelmia ei voida poistaa, mutta osan syntyminen on varmasti estettävissä.
- Ymmärrä, että aina kaikkein energiatehokkaimpaan ratkaisuun ei päädytä liiketoiminnallisista syistä tai käyttäjien tarpeista johtuen.
- Pidennä käyttämiesi laitteiden uusimissykliä. Vältä vaihtamasta täysin toimivaa laitetta uuteen. Korjauta laitteita mahdollisuuksien mukaan.

Näiden lisäksi jo aiemmin mainittu Sustainable Web Design -kirja tarjoaa paljon vinkkejä suunnittelijoille energiatehokkaamman palvelun suunnittelemiseksi.[71]

## 11.3 Testaajille ja laadunvarmistukselle

Laadunvarmistus ja siellä toimivat testaajat ovat sovelluskehittäjien jälkeen seuraavat ihmiset, jotka näkevät ohjelmiston toiminnassa. Ohjelmisto ei ole vielä tässä vaiheessa päätynyt loppukäyttäjille, joten siihen on helpompi tehdä muutoksia. Tästä syystä laadunvarmistus ja siellä suoritettava testaus onkin aitiopaikka varmistaa ohjelmiston energiatehokkuus.

Toimi seuraavasti:

- Ota asia vakavasti ja ryhdy toimeen.
- Lisää ohjelmiston energiatehokkuuteen liittyviä testejä. Hyvä paikka niille on erilaisten rasitustestien ja tehokkuutta mittaavien testien yhteydessä.
- Seuraa ohjelmiston energiankulutuksen muutosta versiosta toiseen. Puutu sovelluskehittäjien kanssa äkillisesti kasvaneeseen kulutukseen välittömästi sen havaittuasi. Vastaavasti välitä tietoa myös muutoksista ajan yli ja käy aktiivista keskustelua energiankulutuksesta.

[71] abookapart.com/products/sustainable-web-design

- Pyri tunnistamaan tässä kirjassa esitettyjä hukkia ohjelmiston testattavassa versiossa ja raportoi ne sovelluskehittäjille tai suunnittelijoille.
- Suunnittele ja rakenna mahdollisimman energiatehokkaat testausympäristöt. Varmista, että niitä käytetään oikein. Pyri käyttämään laitteet tehokkaasti ja maksimoimaan niiden elinikä.
- Sammuta kaikki ympäristöt, kun ne eivät ole käytössä.
- Sijoita raskaat kuormitustestit mahdollisuuksien mukaan halvan energian hetkiin.
- Mieti erilaisille automaattisille testeille järkevä ajotaajuus. Pura käytetty testauskriteeristö useampaan kriittisyystasoon ja mieti, että minkä tason testit tulee ajaa missäkin vaiheessa sovelluskehitysprosessia.
- Jos olet senioritason testaaja, pidä huolta vähemmän kokeneista testaajista ja auta heitä ymmärtämään energiatehokkuuden vaalimista testauksen avulla.

## 11.4 Ohjelmistoyrityksille

Ohjelmistoyrityksillä on suuri valta sovellusten käyttämään energiaan, koska nimensä mukaisesti he sovelluksia toteuttavat. Jotta asia otetaan vakavasti, ylemmän johdon pitää olla asian kanssa sinut ja nähdä liiketoimintamahdollisuus  vihreän koodin tuottamisessa. Toivottavasti tämä kirja on osaltaan auttanut vakuuttamaan, että ekotehokas koodaus on tulevaisuutta.

Toimi seuraavasti:

- Ota asia vakavasti ja ryhdy toimeen. Tänään, ei tulevaisuudessa.
- Kouluta henkilökuntasi tunnistamaan tehokkaan ja tehottoman koodaamisen ero. Tämä kirja auttaa asiassa.

- Katso koko arkkitehtuuria, tunnista oikeat kohdat muutoksille – älä osaoptimoi.
- Varmista, että henkilökuntasi laitteisto on tehokasta suhteessa niiden energiankulutukseen. Samoin maksimoi laitteiden käyttöikä ja varmista niiden asianmukainen kierrätys tai jatkokäyttö.
- Selvitä tuottamiesi järjestelmien energiatehokkuus ja etsi parannuskohteita.
- Pyri hiilineutraaliuteen tai ainakin päästöjen minimointiin koko liiketoiminnassasi.
- Evankelisoi asiasta asiakkaillesi ja yhteistyökumppaneille. Vaadi yhteistyökumppaneiltasi hiiliviisautta. Vaihda tarvittaessa, jos viesti ei tunnu menevän perille.

## 11.5 Ostajille

Ilmastoystävällisten ohjelmistojen ostaminen tulee tulevaisuudessa olemaan selkeästi tärkeämpää ja joskus jopa kriittistä. Omien toimien ja ajattelun muuttaminen kannattaa aloittaa aikaisin, jotta mahdollisen regulaation tullessa ei oltaisi housut kintuissa.

Mikäli ohjelmistojen ostamisessa otettaisiin ilmastonäkökulma huomioon – mitä ei juurikaan tehdä tällä hetkellä – siitä kiittäisivät sekä planeetta että yrityksen CFO. Tehokkaammaksi tehty ohjelmisto kuluttaa vähemmän resursseja, energiaa, palvelimia, siirtoyhteyksiä ja niin edelleen, mikä on suoraa säästöä ohjelmiston ostajalle.

On tärkeä muistaa, että ostajat käyttävät suurta valtaa – kultaisen säännön mukaisesti: “kellä on kulta, se tekee säännöt.”

Toimi seuraavasti:

- Hanki hiiliviisaat digitalisaatiokumppanit – palvelun suunnittelu, toteutus, käyttöpalvelut, datayhteydet, yms. Vaadi heiltä ilmastoa

kunnioittavia toteutuksia ja ota tämä huomioon budjetoinnissasi. Auta heitä etenemään oikeaan suuntaan.

- Mieti, miten digitaaliset ratkaisusi vaikuttavat ympäristöön ja mukauta niitä tarvittaessa. Keskity ensiksi niiden hiilikädenjälkeen ja sitten vasta niiden hiilijalanjälkeen. Mutta katso molemmat.
- Katso koko arvoketjua, älä osaoptimoi. Ota tarvittaessa koko liiketoimintaverkostosi huomioon.
- Älä ahnehdi kaikkea kerralla. Tee muutoksia järkevässä tahdissa, niin jaksat perille.
- Lyö rumpua ja evankelisoi aiheesta.

Lisäksi ostajien osto-organisaatiolle lisävinkkeinä:

- Mieti omalle organisaatiollesi sopivat hankintakriteerit, joilla voidaan arvioida yleisesti vastuullisuutta ja erityisesti hiilineutraaliutta ja vihreää koodaamista.
- Säädä painoarvot uusille ja aiemmille kriteereille organisaation arvojen mukaisesti.
- Jaa kokemuksiasi muille hankkijoille.

## 11.6 Käyttäjille

Sovellusten energiankulutus lähtee tyypillisesti käyttäjästä. Jos sovelluksella ei ole käyttäjiä, se lähinnä odottaa pienellä energiankulutuksella. Omaan käyttöön kannattaa suhtautua uteliaisuudella ja miettiä, että pärjäisinkö vähemmälle. Olisinko jopa onnellisempi, jos kännykkä ei olisi jatkuvasti liimautunut käteeni?

Onko osa toimistani ilottomia ja muistuttaa addiktiota? Pyörinkö jatkuvasti sosiaalisessa mediassa, mutta en oikein saa siitä mitään irti? Tällaisten haitallisten käytösmallien tunnistamisessa ja niiden katkaisussa paranee sekä oma mielenterveys että sovellusten energiankulutus.

Ja viimeisenä ja luultavasti merkittävimpänä vinkkinä: Vähennä videoiden käyttöä. Internet-liikenteestä noin 80 % on videoita ja niiden välittämän tiedon lukeminen tekstinä on merkittävästi tehokkaampaa. Varo sovelluksia, joissa videoita tuputetaan jonossa.

Toimi seuraavasti:

- Tunnista ohjelmistojen kanssa käyttämäsi aika. Erilaiset ruutuaikalaskurit ovat tässä avuksi. Vähennä käyttöä, jos mahdollista. Kaikkein energiatehokkain sovellus on käyttämätön sovellus. Jos jostakin sovelluksesta, esimerkiksi sosiaalisesta mediasta, tulee huono tai riittämätön olo, lopeta sen käyttö ja poista sovellus. Älä palaa.

- Vähennä videoiden katselua. Mieti, voitko lukea tai kertoa saman asian tekstinä videon tai kuvan sijaan. Älä tunge kaikkea viestintää täyteen meemejä ja gif-kuvia, vaan käytä niitä harkiten.

- Harvenna laitteidesi uusintasykliä ja pidä niistä hyvää huolta. Osta sellaisia laitteita, joille luvataan ohjelmisto- ja tietoturvapäivityksiä pitkälle ajalle. Mieti korjauttamista uusimisen sijaan, erityisesti akkujen vaihto voi antaa useita hyviä vuosia puhelimelle tai kannettavalle tietokoneelle.

- Kytke laitteesi mahdollisuuksien mukaan kiinteään verkkoon, erityisesti langaton laajakaista kannattaa vaihtaa kiinteään. Suosi wifiä mobiilitiedonsiirron sijaan, ja niin ikään 5G:tä 4G:n sijaan.

- Säädä laitteidesi näyttö sammumaan aiemmin ja poista näytönsäästäjät – virraton tai tumma ruutu ovat ekologisempia.

- Älä koske bitcoiniin ja vastaaviin energiaa tuhlaaviin ratkaisuihin. Kaikki digitaaliset ratkaisut eivät ole automaattisesti parempia kuin analogiset tai aiemmin käytössä olleet.

- Sovellusten käytön sijasta käytä pienet hukkahetket haaveiluun. Siedä tylsyyttä.

- Ota kaverisi ja tuttavasi mukaan talkoisiin. Älä kuitenkaan painosta tai saarnaa, vaan houkuttele heidät mukaan positiivisuuden kautta.

## Tiivistettynä

1. Energiatehokkaiden ratkaisujen kehittämisessä olennaista on lähestyä energiankulutuksen haasteita yksilöllisesti ja välttää käyttämästä kaikkeen sopivaa yhtä ratkaisua.

2. Kehittäjillä, suunnittelijoilla, testaajilla, ohjelmistoyrityksillä, ja ostajilla on omat roolinsa energiatehokkaan ohjelmiston luonnissa. Kunkin ryhmän tulisi ottaa vastuunsa vakavasti, tehdä yhteistyötä ja osallistua aktiivisesti ilmastoa suojeleviin käytäntöihin.

3. Kaikkien tahojen tulisi suosia energiatehokkaita laitteita ja paneutua niiden elinkaaren pidentämiseen. Laitteiden käytössä tulee noudattaa mahdollisimman vähän energiaa kuluttavia käytäntöjä.

4. Sovellusten tekijöiden tulisi aktiivisesti puhua energiatehokkuuden puolesta, kehittää teollisuudenalojen laajuisia ratkaisuja ja yleisesti omaksua myönteinen ajattelutapa muutokseen.

# 12 Yhteenveto

IT-ratkaisujen kuluttama energia on kasvanut voimakkaasti viimeisten vuosien aikana ja asiaan on havahduttu äskettäin. Ratkaisujen hiilikädenjälki on ollut keskimäärin erittäin positiivinen, eli niiden avulla on voitu tehostaa muita prosesseja päästöttömimmiksi. Ilmastonmuutoksen torjumiseen liittyvien tavoitteiden saavuttamiseksi on kuitenkin katsottava myös IT-ratkaisujen energiankulutusta.

Tällä hetkellä alan trendit vievät valitettavasti väärään suuntaan energiatehokkuuden kannalta ja alaa ei juurikaan säädellä energiankulutuksen suhteen. Nämä tulevat varmasti muuttumaan tulevaisuudessa. Tällä hetkellä ala kerää osaamista vihreän koodin tuottamiseksi ja aiheesta on erilaisia ohjelmia käynnissä. Edistystä on vielä vähän, mutta kehitys on kiihtyvää.

Modernien sovellusten energiankulutus voidaan jakaa karkeasti kolmeen erilliseen osaan: palvelinkeskuksissa toimivien palvelujen ja sisäisen verkkoliikenteen kulutus, tiedonsiirtoon palvelinkeskuksesta käyttäjän päätelaitteelle kuluva energia ja päätelaitteen tiedon käsittelyyn ja näyttämiseen käyttämä energia. Jokainen sovellus on yksilö ja mallin painotukset ja yksityiskohtien runsaus vaihtelevat sovelluksesta toiseen.

Tällä hetkellä mittausdataa sovellusten tehokkuudesta on julkisesti valitettavan vähän tarjolla. Vielä ei ole mahdollista esimerkiksi verrata oman sovelluksen tehokkuutta vastaaviin muihin sovelluksiin tai toimialan keskiarvoon. Samoin tietoteknisten laitteiden energiatehokkuuteen ei ole olemassa vastaavaa standardia kuin esimerkiksi kodinkoneilla.

Energiankulutusta voidaan kuitenkin ryhtyä jo nyt minimoimaan ilman mittausdataa. Sovellusten toimintojen optimointi, tietomäärien minimointi ja turhien asioiden poisto johtavat kaikki parempaan energiatehokkuuteen.

Sovellusten energiankulutusta voidaan jäsentää hukan avulla. Tunnistetaan sovelluksen toiminnasta piirteitä, jotka eivät tuota arvoa vaan kuluttavat ainoastaan energiaa. Näiden hukkien poistolla sovelluksen toiminta tehostuu. Hukkia on useita erilaisia. Tässä kirjassa tunnistetut hukat eivät ole välttämättä täydellinen lista, vaan niiden määrä tulee kasvamaan energiatehokkuuteen liittyvän ymmärryksen ja kokemuksen karttuessa.

Hukkien poistaminen ei ole pelkästään sovelluskehittäjien tehtävä, vaikka osa hukista onkin luonteeltaan teknisiä. Tyypillisesti hukan taustalla on liiketoimintasyyt tai käyttäjille tarjottu kokemus. Näitä asioita ei voida arvioida pelkästään energiankulutuksen kannalta, vaan on tehtävä punnittuja päätöksiä ja kompromisseja.

Tässä kirjassa on esitetty joukko erilaisia ratkaisuja tehokkuuden parantamiseksi ja hukan poistamiseksi. Jälleen kerran on todettava, että ratkaisuja tulee soveltaa sovelluksen kontekstissa ja yksikään ratkaisu ei ole yksinään riittävä. Koska sovellusten tekemisen takana on liiketoimintaa, kaikkea ei voida muuttaa kerralla. Vaikutuksen arviointia varten tässä kirjassa esitellään kaksi erilaista nelikenttää.

Tekoäly, lohkoketjut ja kryptovaluutat, esineiden internet (IoT) sekä data yleisesti lisäävät energiankulutusta ja päästöjä IT-alalla. Yleisesti energiankulutuksen hallitseminen näillä nousevilla teknologioilla on ratkaisevaa kestävien IT-alan käytäntöjen varmistamiseksi ja ympäristövaikutusten vähentämiseksi.

Sovelluksen energiatehokkuuden lisäksi kannattaa myös laskea ja minimoida sovelluskehitystyön hiilijalanjälki. Tätä varten Koodia Suomesta ry. on julkaissut hiilineutraaliusmerkin. Merkin kriteeristöä noudattamalla saa hyvät toimintaohjeet oman hiilijalanjäljen käsittelyyn, vaikka itse merkkiä ei olisikaan hakemassa.

Eri toimijoille – sovelluskehittäjät, suunnittelijat, testaajat, käyttäjät, ohjelmistoyritykset, asiakasyritykset ja heidän ammattiostajansa – on listattu omat suositukset. Olennaista kaikissa on, että nyt pitää ryhtyä toimeen per heti.

# 13 Kiitokset

Tämän kirjan kirjoittaminen on pyörinyt mielessäni jo jonkin aikaa, ensiksi muodottomina ja lopulta analyyttisempina ajatuksina. Lopulta kirja oli pakko kirjoittaa, koska ajatus siitä ei enää antanut minulle rauhaa.

Aloin kiinnostumaan vihreästä koodaamisesta Koodia Suomesta ry:n hallituksessa käytyjen keskustelujen pohjalta. Yhdessä hallituksen jäsenen, Hiottu Oy:n toimitusjohtaja Satu Lapinlammen kanssa aloimme selvittämään tilannetta. Satun iloinen ja silti erittäin päämäärätietoinen tyyli viedä ympäristöasioita eteenpäin on ollut energisoivaa seurattavaa. Ilman häntä en olisi tällä matkalla.

Ensimmäisen erinomaisen yhteenvedon meille Satun kanssa tarjosi Sitran asiantuntija Lotta Toivonen ja Lotan kanssa olemme vaihtaneet ajatuksia useamman kerran prosessin aikana. Toinen merkittävä askel eteenpäin oli Tietoyhteiskunnan kehittämiskeskus Tieken järjestämä seminaari, jossa ensimmäisen kerran pääsin asiasta puhumaan ja tutustumaan muihin samoja ajatuksia pohtiviin tekijöihin. Suuret kiitokset tästä Tieken toiminnanjohtaja Hanna Niemi-Hugaertsille.

Näissä yhteyksissä tutustuin Aalto-yliopiston professori Jukka Manneriin, jonka ajatukset yksinkertaisemmista ja siten tehokkaimmista ratkaisuista resonoivat todella voimakkaasti. Samoin LUT-yliopiston professori Jari Porras on ollut valtavan tärkeä keskustelukumppani näiden asioiden tiimoilta.

Päädyin lopulta Tieken Green ICT -hankkeen ohjausryhmään ja sen puheenjohtajaksi. Sain onnekseni tutustua projektipäällikkö Antti

Sipilään, joka on ollut arvokas apu monissa keskusteluissa ja linkityksissä alan osaajiin.

Rukakeskuksen vastuullisuusjohtaja Jusu Toivonen on toiminut inspiraationa siinä, kuinka ympäristöön liittyviä asioita voidaan parantaa merkittävästi määrätietoisella ja suunnitelmallisella asenteella ja sitkeydellä. Hänen kanssaan käydyt keskustelut ovat valaneet uskoa, että olen oikealla tiellä.

Exoven vastuullisuudesta vastanneen Anna Savisaaren kanssa pallottelin erilaisia ideoita ja kehitin omaa osaamistani vastuullisuuden saralla. Ilman näitä pohdintojamme en olisi tätä kirjaa koskaan tullut kirjoittaneeksi.

Keskusteluni aiheesta Exoven myynti- ja markkinointijohtajan Päivikki Kyykän kanssa johtivat ajatukseen hukasta. Tämä ajattelu avasi viimeisen lukon – olin pitkään pohtinut, että miten asiaa voi viedä analyyttisesti eteenpäin ilman mittausdataa. Leanista lainattu hukka tarjosi vapauttavan ajattelumallin edistää koodin ekotehokkuutta ilman jatkuvaa mittausta.

Erityiskiitokset tämän kirjan esilukijoille, joilta sain merkittävän määrän uusia ajatuksia ja huomioita lisättäväksi tekstiin. Esilukijoina toimivat Valohain CTO Aarni Koskela, Exoven CTO Kalle Varisvirta, Rebasen vanhempi ohjelmistoinsinööri Tommi Sinivuo ja Exove Designin suunnittelija Katri Pakula. Lisäksi Exoven Growth Marketer Essi Rostedt oikoluki tekstin ja antoi sille kielenhuoltoa. Vastaavasti toisen laitoksen oikoluvusta ja kielenhuollosta vastasi Päivikki Kyykkä ja Exoven Growth Marketer Jessica Holmberg. Toisen laitoksen esilukijoina toimivat Kalle Varisvirta ja TietoEvryn Senior Software Architect Tommi Lehtinen. Ilman teitä tämä kirja olisi huomattavasti suppeampi ja ylipäänsä heikompi laadultaan.

Sanomattakin lienee selvää, että kaikki mahdolliset virheet ovat allekirjoittaneen käsialaa.

# 14 Palaute

Otan mielelläni vastaan palautetta ja uusia ideoita vihreämmän koodin tekemiseen. Aiheella on merkitystä sekä IT-alan kehittymisen että ilmastonmuutoksen hidastamisen suhteen. Lisäksi se on minulle henkilökohtaisesti hyvin tärkeä ja siksi pyrin parantamaan tämän kirjan laatua ja merkityksellisyyttä.

Haluan tietää, jos koit jonkun aiheen jääneen käsittelemättä tai käsittelyn olleen vain pintapuolista. Samoin kuulen mielelläni uusia ideoita hukista tai vinkkejä ohjelmistojen tehokkuuden parantamiseksi. Sanomattakin on selvää, että kaikki virheet olisi syytä raportoida.

Tähän laitokseen on annetun palautteen pohjalta kirjoitettu testaamisesta ja tekoälystä. Lisäksi muutama huomaamanne kirjoitusvirhe on korjattu. Kiitos kaikille palautetta antaneille!

Jos haluat parantaa kirjani laatua tai vaikkapa vain kehua sitä, niin toivoisin sinun täyttävän lyhyen kyselyn seuraavan linkin takaa:

bit.ly/vihreakoodipalaute

Kiitos.

# Exove ja UpCloud tuovat markkinoille hiilineutraalit verkkopalvelut

Exove ja UpCloud ovat päättäneet yhdistää voimansa vastuullisten pilvipohjaisten verkkopalvelutoteutusten tarjoamiseksi paikallisesti tai kansainvälisesti toimiville yrityksille. Yhteistyö tarjoaa asiakkaille luotettavan ja nykyaikaisen tavan tehdä sisällönhallintaa sekä verkkopalveluja kustannustehokkaasti ja laadukkaasti ympäristöä säästäen.

Yhteistyö huomioi paitsi asiakkaan tarpeet myös ympäristön ja yhteiskunnan. Vastuullisuuden suhteen korostuvat etenkin hiilijalanjäljen ja verkkopalveluiden ympäristökuormituksen minimointi. Olemme kehittäneet yhdessä CO2-laskurin, jonka avulla pystytään seuraamaan verkko- ja pilvipalveluiden päästöjä lähes reaaliaikaisesti. Merkittävintä on kyky mitata ja vähentää verkkopalvelun hiilijalanjälkeä koko ketjun laajuudelta.

UpCloudin pilvipalvelut ovat saatavilla 12 konesalissa maailmanlaajuisesti ja näistä suomessa sijaitsee kaksi. Yritys on globaaleilla pilvimarkkinoilla tunnettu korkeasta saatavuudesta ja suorituskyvystä. UpCloud tarjoaa asiakkailleen 99,99% palvelulupauksen (SLA:n) ja markkinoiden parhaimman suorituskyvyn ja 24/7 tuki-palvelun.

*Olemme UpCloudilla nostaneet vastuullisuuden ja kestävän kehityksen toimintamme keskiöön ja osana tätä pyrimme myös neutralisoimaan hiilijalanjälkemme pidemmällä aikavälillä.*

*Exove on kuitenkin vienyt kaiken tämän entistä pidemmälle verkkopalveluiden osalta ja olemme innoissamme tästä kehittyvästä yhteistyöstä.*

**Antti Vilpponen**,
UpCloud

**Jos kiinnostuit, ota yhteyttä**
***business@exove.com***

# Haluatko oman organisaatiosi koodin vihertämään?

Exove tarjoaa koulutusta ja valmennusta ohjelmistokehittäjille ja muulle organisaatiolle sekä ylemmän johdon konsultointia energiatehokkaiden ja hiilineutraalien IT-järjestelmien toteuttamiseksi. Auditoimme järjestelmiä ja analysoimme yrityksen ICT-kokonaisarkkitehtuureja.

## Koulutus

Koulutamme sovelluskehittäjiä, suunnittelijoita ja arkkitehteja. Koulutuksessa käsitellään vihreän koodaamisen perusperiaatteet, hukkien tunnistaminen ja järjestelmän järkevä optimointi.

## Konsultointi

Konsultoimme yrityksen johtoa vihreään koodiin liittyvissä erityiskysymyksissä, autamme energiatehokkuuden kehittämisessä ja organisaation kyvykkyyden kasvattamisessa.

## Arkkitehtuurianalyysi

Analysoimme ICT-kokonaisarkkitehtuurin energiatehokkuuden kannalta ja laadimme roadmapin tilanteen parantamiseksi. Koulutamme tarvittaessa järjestelmien toimittajat.

## Auditointi

Auditoimme yrityksen nykyiset järjestelmät energiatehokkuuden suhteen ja laadimme ehdotukset parannuksista sekä arvion säästöistä.

## Valmennus

Autamme viemään vihreän koodaamisen muutoksen organisaatiossanne valmentamalla ja ohjaamalla avaintekijöitänne.

**Jos kiinnostuit, ota yhteyttä**
***business@exove.com***