LAB University of Applied Sciences

Nimikehallinnan laadunvarmistuksen tehostaminen

LAB-ammattikorkeakoulu
Insinööri (ylempi AMK), IoT:stä tekoälyyn
2024
Ekaterina Ruotsalainen

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Ekaterina Ruotsalainen	Opinnäytetyö, YAMK	2024
	Sivumäärä	
	69	

Työn nimi

Nimikehallinnan laadunvarmistuksen tehostaminen

Tutkinto ja koulutusala

Insinööri (YAMK), IoT:stä tekoälyyn

Toimeksiantajaorganisaatio

Tiivistelmä

Viime vuosikymmenien aikana tietojen digitalisaation myötä yritysten nimikedata on kasvanut räjähdysmäisesti, ja siitä on tullut vaikeasti hallittava. Huonolaatuinen nimikedata voi olla este yrityksen sujuvalle liiketoiminnalle. Yleisimmät nimikehallinnan haasteet ovat muun muassa virheellinen tai puuttuva data, nimikeduplikaatit sekä vanhettuneet nimikkeet. Opinnäytetyössä käsiteltiin nimikedatalaadun hallinta master datan näkökulmasta sekä pohdittiin yritysten nimikedatan laatuun liittyvistä ongelmista ja keinoista, joilla nimikedata voitaisiin huoltaa.

Työn käytännön osuudessa etsittiin ratkaisut nimikeduplikaattien sekä virheellisten nimikeattribuuttien löytämiseen nimikedatasta. Nimikeduplikaattien hakua suoritettiin sekä perus Excel-toiminnallisuuksia hyödyntäen että koneoppimismallin avulla Python koodausta käyttäen. Nimikeattribuuttien tarkistukseen oli kehitetty oma työkalu, jossa tarkistustoiminnot suoritettiin VBA-makrolla käynnistettävän Python skriptin avulla.

Asiasanat

Nimikehallinta, master data, datan laatu

Abstract

Author(s)	Type of Publication	Published
Ekaterina Ruotsalainen	Master's thesis	2024
	Number of Pages	
	69	

Title of Publication

Item management quality control enhancement

Degree, Field of Study

Master of Engineering, From IoT to AI

Organization of the client

Abstract

Due to data digitalization over the last few decades the volume of companies' item data has grown exponentially, and item data has become decentralized and difficultly manageable. Poor item data quality can be an obstacle to the company's business smooth functioning. The most common item management challenges include inaccurate or missing data, item duplicates and outdated items. The thesis considers item data management from the perspective of master data and companies' challenges associated with item data quality maintenance.

Solutions to find item duplicates and inaccurate item attributes from the item data were developed in the research chapter of the thesis. Item duplicates search was performed both using basic Excel functionalities and with the help of a machine learning model using Python coding. For finding incorrect item attributes it was developed a check tool whose functions were performed in a Python script run from Excel using VBA macro.

Keywords

Item management, master data, data quality

Sisällys

Lyhenteet ja käsitteet

BOM (Bill of Material) - Asiakirja tai luettelo, joka yksityiskohtaisesti määrittelee tuotteen tai tuotteen valmistukseen tarvittavat komponentit ja niiden suhteet. BOM on tärkeä osa tuotannon ja valmistuksen hallintaa, se auttaa organisaatioita pitämään kirjaa tarvittavista materiaaleista, komponenteista ja työvaiheista tuotteen valmistamiseksi.

CAD (Computer-Aided Design) - Tietokoneavusteinen suunnittelumenetelmä, joka käyttää tietokonetta suunnitteluprosessissa monimutkaisten 2D- ja 3D-mallien luomiseen, muokkaamiseen ja analysointiin esimerkiksi insinööritieteissä, arkkitehtuurissa ja tuotesuunnittelussa.

DAM (Data Asset Management) - Digitaalisten aineistojen hallintajärjestelmä, joka auttaa organisaatioita tallentamaan, järjestämään, hallitsemaan ja jakamaan digitaalisia resursseja, kuten kuvia, videoita, äänitiedostoja ja asiakirjoja. DAM-järjestelmä helpottaa digitaalisten aineistojen löytämistä, käyttöä ja jakamista organisaatiossa.

DSR (Data Science Resesrch) - Tutkimusmenetelmä, jota käytetään muun muassa tietojärjestelmien ja tietotekniikan aloilla. Se keskittyy uusien ratkaisujen suunnitteluun ja kehittämiseen käytännön ongelmien ratkaisemiseksi. DSR:ssä luodaan konkreettisia "artefakteja" (esimerkiksi ohjelmistoja) ja arvioidaan niiden toimivuutta. Tavoitteena on sekä käytännön ongelmien ratkaiseminen että teoreettisen tiedon edistäminen.

ERP (Enterprise Resource Planning) - Liiketoimintajärjestelmä, joka auttaa organisaatioita hallitsemaan ja integroimaan eri liiketoimintaprosessejaan yhtenäisessä tietojärjestelmässä. Se kattaa yrityksen useat toiminnot, kuten taloushallinnon, varastonhallinnan, myynnin, hankinnan ja resurssienhallinnan. ERP-järjestelmät tarjoavat organisaatioille reaaliaikaista tietoa, automatisoituja prosesseja ja parantavat yhteistyötä eri osastojen välillä.

Master data - Yrityksen toiminnan kannalta välttämätöntä ydintietoa, joka on jaettu läpi organisaation ja pysy muuttumattomana riippumatta siitä, missä yrityksen yksikössä sitä käytetään. Se on keskeinen osa tietojenhallintaa ja auttaa organisaatiota ylläpitämään ja hallitsemaan yhtenäisiä ja laadukkaita tietoja, joita tarvitaan liiketoiminnan prosesseissa ja päätöksenteossa.

MDM (Master Data Management) - Prosessi ja tekniikka, jonka avulla organisaatiot hallitsevat, ylläpitävät ja varmistavat yrityksen keskeisten tietojen, eli master datan yhdenmukaisuuden eri liiketoimintajärjestelmissään. MDM pyrkii varmistamaan, että nämä keskeiset

tiedot ovat tarkkoja, yhtenäisiä ja helposti saatavilla koko organisaatiossa, mikä parantaa päätöksentekoa ja liiketoiminnan tehokkuutta.

NLP (Natural Language Processing) - Tietotekniikan ala, joka keskittyy tietokoneiden kykyyn ymmärtää, tulkita ja käsitellä ihmisen kirjoittamaa tai puhuttua kieltä. Se hyödyntää koneoppimisen ja tekoälyn tekniikoita mahdollistaakseen tehokkaan vuorovaikutuksen ihmisen ja tietokoneen välillä kielen avulla.

PDM (Product Data Management) - Tuotetietojen hallintajärjestelmä, joka auttaa organisaatioita hallinnoimaan tuotteisiin liittyviä tietoja ja tiedostoja kuten nimikkeitä, piirustuksia ja tuotedokumentaatiota. PDM-järjestelmät ovat erityisen hyödyllisiä tuotekehityksessä ja suunnitteluprosesseissa, sillä ne auttavat organisaatioita pitämään tuotetiedot ajan tasalla ja helposti jaettavina eri tiimien kesken.

PDSA (Plan-Do-Study-Act) - Oppimisen ja tiedon hankkimisen järjestelmällinen prosessi tuotteen, menetelmän tai palvelun jatkuvaa parantamista varten. Tämä integroitu oppimis- ja kehitysmalli tunnetaan myös nimellä Demingin laatuympyrä.

PLM (Product Lifecycle Management) - Tuotteen elinkaaren hallintajärjestelmä, joka auttaa organisaatioita suunnittelemaan, kehittämään, hallitsemaan ja seuraamaan tuotteen koko elinkaarta sen suunnittelusta valmistukseen ja ylläpitoon asti. PLM-järjestelmät mahdollistavat tuotetiedon hallinnan, yhteistyön tiimien välillä, ja ne auttavat parantamaan tuotteen laadunhallintaa ja kustannustehokkuutta.

PIM (Product information Management) - Tuotetietojen hallintajärjestelmä, joka keskittyy tuotetiedon keräämiseen, ylläpitoon ja jakeluun organisaatiossa. PIM-järjestelmät auttavat organisaatioita hallitsemaan ja yhtenäistämään tuotteisiin liittyviä tietoja, kuten kuvia, kuvauksia, hintoja ja teknisiä tietoja, ja mahdollistavat niiden tehokkaan jakamisen eri myynti- ja markkinointikanaviin, kuten verkkokauppaan ja katalogeihin. Tämä auttaa parantamaan tuotteiden laadunhallintaa ja tehostamaan markkinointiprosesseja.

POC (Proof of Concept) - Käytännön koe tai näyttö, jonka tarkoituksena on testata ja osoittaa, että jokin idea, konsepti tai teknologinen ratkaisu on toteuttamiskelpoinen ja toimii odotetulla tavalla. POC:n avulla pyritään vahvistamaan, että suunniteltu projekti tai innovaatio voi toimia käytännössä ennen kuin siihen sijoitetaan resursseja ja aikaa. Se on yleinen käytäntö erityisesti teknologiaprojekteissa ja uusien liikeideoiden kehityksessä. POC voi olla

esimerkiksi prototyyppi, koeajo, pienimuotoinen testi tai muu näyttö siitä, että idea tai konsepti toimii tosielämässä.

Regular Expressions (Regex) eli säännölliset lausekkeet – Merkkijonojen hakuun ja muokkaamiseen käytettäviä matemaattisia ilmaisuja. Ne tarjoavat tehokkaan tavan määritellä hakukriteerejä, kuten tietyn merkkijonon kaava tai rakennemalli. Regexit ovat laajalti käytössä ohjelmoinnissa, tekstinkäsittelyssä ja tiedon etsinnässä, ja ne voivat auttaa tunnistamaan, muokkaamaan tai eristämään tiettyjä merkkijonoja suuresta määrästä tekstiä.

RNN (Recurrent Neural Network) – Keinotekoinen neuroverkko, joka on suunniteltu käsittämään sekvenssejä ja aikasarjoja. Toisin kuin perinteiset neuroverkot, RNN:llä on kyky ottaa huomioon aikaisemmat tapahtumat ja tilat sekä käsitellä syötteitä järjestyksessä. Tämä mahdollistaa verkkojen käytön erilaisten aikasarjojen ja sekvenssien käsittelyyn, kuten kielimallinnukseen, aikasarjaennustamiseen ja käsin kirjoitetun tekstinsyötön tunnistamiseen.

Tietue – datan hallintamielessä yksi itsenäinen looginen tietokokonaisuus järjestelmässä. Tietueella on yleensä joku yksilöivä avaintieto, esimerkiksi järjestelmän antama tunniste sekä tietyt ominaisuudet (kentät), jotka sisältävät tietoja kyseisestä yksiköstä. Relaatiotietokannassa yhtä tietuetta edustaa yksi rivi.

1 Johdanto

1.1 Työn tausta ja tavoitteet

Viime vuosikymmenien aikana tietojen digitalisaation myötä lähes kaikkien yritysten tuotetieto on siirtynyt digitaaliseen muotoon ja sitä hallitaan erilaisten tietojärjestelmien avulla. Reilussa 20 vuodessa tuotetiedon määrä on kasvanut räjähdysmäisesti. Vuosien saatossa monissa yrityksissä havaittiin samankaltaiset ongelmat: datamassasta tuli valtava ja vaikeasti hallittava, hajautetun tiedon hallinnan vuoksi datan laatu on heikentynyt niin, että siitä usein syntyy esteitä joustavalle liiketoiminnalle.

Toimiva nimikehallinta on yritysten tuotetiedon hallinnan perusta. Ilman nimikkeiden käsittelyyn liittyviä selkeitä ja yhtenäisiä prosesseja, tuotetiedon hallinta on tehoton. (Peltonen ym. 2002, 14.) Yritysten ja tuotetiedon kasvun myötä syntyy tarve huoltaa ja ylläpitää nimikkeisiin liittyvää master dataa.

Opinnäytetyön toimeksiantaja on yritys, joka tuottaa muun muassa nimikehallintaan liittyviä tehtäviä osana suunnittelupalveluita tai erikseen nimikehallintapalveluina. Tämän työn tavoite on etsiä ja luoda ratkaisuja, joiden avulla yrityksen nimikehallinnan laadun voisi varmistaa ja parantaa. Luodut ratkaisut ja työkalut mahdollistaisivat nimikehallinnan parantamisen erityyppisissä yrityksissä ja jatkossa työn toimeksiantaja voisi tarjota sitä asiakkailleen esimerkiksi palvelumuodossa.

Työn teoriaosuudessa kerrotaan nimikkeistä ja nimikehallinnan työkaluista yleisesti sekä pohditaan, miksi yritysten master datan laatu huononee, mitkä ovat nimikehallinnan keksiset haasteet ja miten datan laatua voisi parantaa. Käytännön osuudessa yritetään löytää toimivat ratkaisut nimikedatan laadun parantamiseen. Ratkaisujen toimivuutta testataan asiakasyrityksiltä saaduilla nimikedataseteillä.

1.2 Tutkimuskysymykset ja rajaus

Opinnäytetyössä pyritään vastamaan seuraaviin kysymyksiin:

1. Mitkä ovat yrityksen keskeisiä nimikehallintaan liittyviä haasteita?
2. Millä tavalla nimikkeisiin liittyvää master dataa pitää huoltaa?
3. Mitä menetelmiä, teknologioita ja välineitä voidaan käyttää master datan laadun varmistuksessa?

Työssä käsitellään fyysisiä nimikkeitä kuten komponentit ja materiaalit, jättäen esimerkiksi palvelunimikkeet, piirustukset ja muut dokumentit tutkimuksen ulkopuolelle. Nimikehallintaan käytetyt tietojärjestelmät työssä rajataan PDM/PLM- ja ERP-järjestelmiin.

Opinnäytetyön teoriaosuudessa selvitetään nimikehallinnan haasteet sekä toimintatavat, joilla voidaan ehkäistä virheellisten nimikkeiden luomista. Kuitenkin työn painopiste on yrityksen olemassa olevan nimikedatan laadun parantaminen. Tutkimusosuudessa käydään läpi kaksi tyypillistä nimikehallintaan liittyvää ongelmatapausta, jotka ovat nimikkeiden kaksoiskappaleet (duplikaatit) sekä puutteellinen nimikedata.

1.3 Tutkimusmenetelmät

Opinnäytetyön tutkimusmenetelmä on suunnittelututkimus eli Design Science Research (DSR). Menetelmän tavoitteena on löytää ongelman ratkaisu ongelman selittämisen tai ymmärtämisen sijaan (Iivari & Venable 2009, Pello 2018 mukaan). Menetelmä hyödyntää saatua tietoa ongelmien ratkaisemiseksi tai olemassa olevien ratkaisujen parantamiseksi ja samalla luo uutta tietoa tai näkemystä (Baskerville ym. 2015, Pello 2018 mukaan). Lyhyesti DSR-menetelmä voi jakaa kolmen vaiheeseen:

- ongelman, kontekstin ja toimintojen tutkiminen sekä hypoteesin asettaminen
- ratkaisujen suunnittelu ja testaus
- hypoteesin vahvistaminen, tutkimuksen validointi ja yleistäminen muihin sovelluksiin (Horváth 2007, Pello 2018 mukaan).

Peffers ym. (2007) mukaan DSR-prosessi yleensä sisältää seuraavat toiminnot:

- tutkimusongelman tunnistaminen ja määritteleminen sekä ratkaisun arvon perustelu
- ratkaisun tavoitteiden määrittely
- artefaktien suunnittelu ja kehittäminen (rakenteet, mallit, menetelmät jne.)
- ratkaisun esittely käytettyjen artefaktien avulla
- ratkaisun arviointi vertaamalla tavoitteita ja todellisia havaittuja tuloksia artefaktin käytöstä
- ongelman, artefaktin, sen hyödyllisyyden ja tehokkuuden tiedottaminen muille tutkijoille ja ammattilaisille (Pello 2018).

DSR-projektin tulos on aina tarkoituksenmukainen artefakti kuten tuote tai prosessi; se voi olla teknologia, työkalu, metodologia, prosessi, näiden yhdistelmä tai mikä tahansa muu keino tutkimuksen tarkoituksen saavuttamiseksi (Venable & Baskerville 2012, 142, Pello 2018 mukaan).

DSR menetelmän kannalta tämän työn tutkimusongelmana on yrityksen nimikedatan laadun varmistaminen ja parantaminen. Tutkimuksen tavoitteena on luoda ratkaisut, joiden avulla ongelmatapaukset havaitaan ja nimikedata parannetaan. Tutkimuksen artefaktina on nimikedatapaketit, joilla on tarkoitus testata ja esitellä ratkaisun toimivuutta. Työn

lopputuloksena olisi menetelmän onnistunut ajo ainakin yhdellä nimikedatapaketilla (Proof of concept).

2 Nimikehallinta ja sen keskeiset haasteet

2.1 Nimikkeet

Nimikkeellä käsitellään mikä tahansa tuotetiedon hallinnan itsenäinen yksikkö, jolla on identiteetti. Nimikkeinä voidaan esittää kaikki yrityksen liiketoiminnan kannalta merkittävät elementit kuten esimerkiksi tuotteet, materiaalit, palvelut ja dokumentit. (Peltonen ym. 2002, 15, 45.)

2.1.1 Nimiketyypit

Käyttötarkoituksen mukaan ja yritystoiminnasta riippuen nimikkeet voisi jakaa eri tyyppeihin (taulukko 1). Taulukossa esitetyt nimiketyypit ovat esimerkkityyppejä, kaikissa yrityksessä ei välttämättä tarvita kaikkia alamainittuja nimikkeitä, eikä kaikkia luettelon käsitteitä tarvitse aina nimikkeistää.

Fyysiset nimikkeet	**Immateriaaliset nimikkeet**
• Yksittäiset osat ja komponentit	• Palvelut
• Kokoonpanot (muun muassa valmiit tuotteet)	• Toiminnot
• Ostokomponentit (tai materiaalit)	• Asiakkaat
• Suunnittelunimikkeet (muun muassa itsevalmisteiset osat)	• Toimittajat
• Varastoitavat nimikkeet	
• Projektiluontoisetnimikkeet	
• Varaosat	

Taulukko 1.Esimerkit nimiketyypeistä (Peltonen ym. 2002, 15, muokattu)

Myös nimikkeet voi jakaa elinkaaren kannalta aktiivisiin ja käytöstä poistettuihin. Vaikka nimikettä ei käytetä enää, se on silti osaa yrityksen nimikkeistöä ja usein on kytköksessä aktiivisiin tuotteisin, sisältäen tärkeitä tietoa (esimerkiksi korvaavasta tuotteesta).

2.1.2 Nimikeidentiteetti ja attribuutit

Nimikeidentiteetti koostuu identifioivasta tunnisteesta sekä vapaamuotoisesta nimikekuvauksesta. Nimiketunniste yleensä kutsutaan nimikekoodiksi. Nimikekoodi on määrämuotoinen ja yleensä koostuu numeroista tai numeroiden ja kirjainten yhdistelmästä. Nimikekoodi voi olla luokitteleva, eli tunniste kertoo tietyistä nimikkeen ominaisuuksista (esimerkiksi tuoteryhmästä) tai luokittelematon, esimerkiksi juokseva numerosarja, joka ei sisällä minkälaista tuotetietoa. Luokittelevassa nimikekoodissa voi olla omat haittansa. Esimerkiksi nimikeominaisuuksien muutokset tai nimikkeen käytön laajentaminen toiselle toimintayksikölle voivat aiheuttaa ongelmia ja rajoittavat nimikkeen elinkaarta. (Peltonen ym. 2002, 17.) Joten isoimmissa kansainvälisesti toimivissa yrityksissä on suotava käyttää joustavampia, luokittelemattomia nimikekoodeja.

Nimikekuvaus on vapaamuotoista tekstiä, jossa voi olla esimerkiksi tuotteen nimi ja spesifikaatio. Monissa kansainvälisissä yrityksissä käytetään kuvauksen eri kieliversioita. Kuvauskentän vapaamuotoisuus mahdollistaa epätarkat tai virheelliset nimeämistavat, siksi yrityksellä pitäisi olla selkeät nimeämissäännöt sekä sanastot, jos käytössä on monikieliset kuvaukset. (Peltonen ym. 2002, 17.)

Nimikekoodin ja kuvauksen lisäksi nimikkeillä on joukko luokittelutietoja eli attribuutteja. Nimikeattribuutit ovat osa nimikkeen metadataa ja auttavat luomaan täydellisen tuotekuvan (Dassault Systems 2020). Osa attribuuteista on pakollisia, kuten esimerkiksi yksikkö, toiset attribuuttikentät voi jättää tyhjiksi. Eri nimiketyypeillä voi olla erilaiset attribuutit, esimerkiksi tietyt piirustusten attribuutit eroavat komponenttien attribuuteista. Eri attribuuteilla on eri arvotyypit: osa on numeerista dataa (esimerkiksi paino), osa mielivaltaista tekstiä (esimerkiksi valmistaja), tietyt attribuutit on esimääritelty ja ne tulee valita alasvetovalikosta (esimerkiksi materiaaliryhmä) (Peltonen ym. 2002, 26). Nimikkeen identiteetti sekä attribuutit muodostavat nimikkeen näkymän yrityksen tietojärjestelmissä, niiden perusteella nimikkeitä pystytään hakemaan ja käyttämään.

2.2 Nimikehallinta master datan ja tuotetiedon hallinnan osana

2.2.1 Master datan hallinta

Master data on yrityksen toiminnan kannalta välttämätöntä ydintietoa, joka on jaettu läpi organisaation ja pysyy muuttumattomana riippumatta siitä, missä yrityksen yksikössä sitä käytetään (Väre 2019, 23; Fitzsimons 2022). Master data esittää organisaation datana. Siksi master datan hallinnasta hyötyy koko organisaation toiminta ja master datan parantaminen parantaa organisaatiota. Kaikki yrityksen data ei kuitenkaan ole master dataa. Master

datan ominaisuuksista voi päättää URUT-säännön avulla. Sen mukaan master datan attribuuttien ominaisuudet ovat uniikkeja, raportoittua, usein käytettyjä ja tärkeitä. Uniikit datan attribuutit osoittavat, että tietue on tunnistettava ja ainutlaatuinen, niiden muodostama datakokonaisuus ei saa esiintyä samanlaisena kahta tai useampaa kertaa missään yrityksen järjestelmissä (esimerkiksi nimikekoodit). Raportoidut attribuutit kertovat yrityksen tapahtumista, tuloksista ja ohjaavat yrityksen toimintaa. Raportointijärjestelmissä niitä kutsutaan dimensioiksi tai ulottuvuuksiksi, toisin sanoen ne eivät kerro, mitä raportoidaan tai millä tavalla (esimerkiksi tuoteryhmät). Usein käytetyt attribuutit esiintyvät eri organisaation prosesseissa eri osastoilla, eri järjestelmissä. Siksi on kustannustehokkaasta, että tietoa jaetaan läpi koko organisaation. Jos jokainen sidosryhmä olisi itsenäisesti luomassa tätä tietoa omiin järjestelmiin, se veisi ylimääräistä aikaa ja resursseja sekä mahdollistaisi ristiriitaisen tai virheellisen datan syntymisen. Tärkeät attribuutit ovat oleellisia organisaation päivittäisen toiminnan kannalta, sillä tietyt toiminnot sekä prosessien automatisointi eivät onnistu ilman kyseistä tietoa. Nämä tiedot ovat toimintakohtaisia, esimerkiksi tuotteen paino ja mitat eivät välttämättä ole oleellinen tieto tavaratalossa, mutta on pakollinen logistiikkatoiminnoissa. (Väre 2019, 33–36, 45.)

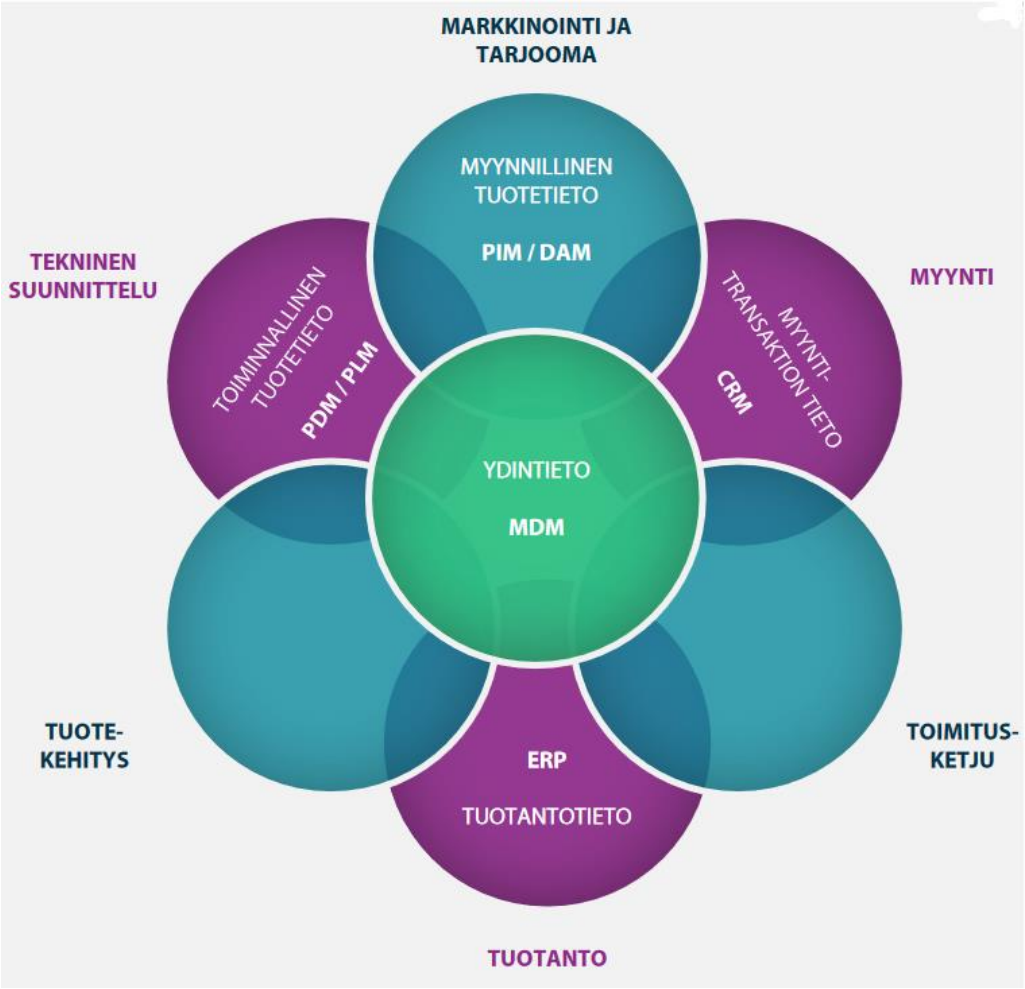
Master data kokoaa yhteen yrityksen liiketoiminnan ja on hyödynnettävissä eri prosesseissa ja järjestelmissä. Näistä prosesseista ja järjestelmistä muodostuu master datahallinta (MDM) (Palatz & Costiander 2019). Master datan hallinnan prosesseilla tarkoitetaan prosessit, joissa master dataa luodaan, muokataan tai ylläpidetään, poistetaan tai arkistoidaan, siirretään järjestelmien välissä (integraatiot), siirretään vanhoista järjestelmistä uusiin (migraatiot), yhdistetään muihin tietoihin analysointia tai raportointia varten. Näiden prosessien avulla täytetään master datan hallinnan perustavoite, joka voidaan tiivistää: kerran tehty - käytettävissä kaikilla. Käytännössä tämä tarkoittaa, että dataa mahdollisimman vähän käsitellään manuaalisesti ja sen saattavuus eri järjestelmiin on varmistettu mahdollisimman tehokkaasti. Kun tuote on kerran luotu tuotekehityksessä, sitä ei tarvitse perustaa uudestaan laskutus- tai varastointi järjestelmiin, vaan kaikki tarvittava tuotetieto olisi valmiiksi saatavissa siinä, missä sitä tarvitaan. (Väre 2019, 89.)

Master datan hallinaan pääperiaatteet ovat:

- Vain yksi tietue vastaa jokaista tosielämän kohdetta organisaatiossa.
- Data luodaan vain kerran yhteen paikkaan, mutta käytetään usein.
- Data päivitetään vain kerran ja varmistetaan, että se on päivitetty kaikkialla.
- Data on saatavilla aina silloin ja missä sitä tarvitaan. (Väre 2019, 37.)

Master datan hallinnan ympärillä ovat eri toimintojen prosessit, kuten tuotekehityksen käyttämät tuotetiedostojen hallinta (PDM) ja tuotteen elinkaaren hallinta (PLM) sekä

markkinoinnin käyttämät tuotetiedonhallinta (PIM) ja aineistohallinta (DAM) (Palatz & Costiander 2019). Yrityksen prosessit toteutetaan eri tietojärjestelmien avulla. Kuvio 1 esittää yrityksen hallussa oleva tietoa, järjestelmiä ja niiden suhdetta toisiinsa.



Kuvio 1. Yrityksen data ja järjestelmät (Palatz & Costiander 2019)

Tuotetiedon hallinta pitkälti keskittyy PDM/ PLM ja ERP-järjestelmiin. Yrityksellä pitäisi olla yhtenäinen tapa hallita tuotetietojansa sekä varmistaa niiden laatu ja ajantasaisuus. On tärkeitä tunnistaa mitä tietoa missäkin prosessissa tarvitaan ja missä järjestelmässä sitä ylläpidetään. (Palatz & Costiander 2019.)

Nimikehallinta on tuotetiedon hallinnan peruspilari (Peltonen ym. 2002, 45). Nimikkeet ovat osa yrityksen master dataa ja luovat pohjan tiimien, organisaatioiden ja sidosryhmien tehokkaalle ja tuotettavalle toiminnalle. Elinkaarensa aikana nimikkeet mahdollistavat muun muassa tuotteiden oston, myynnin, valmistamisen ja huollon.

Keskisuuressa valmistavassa yrityksessä voi olla satoja tuhansia nimikkeitä, jokaiseen niistää liittyy lukuisia attribuutteja. Näin suuren tietomassan hallintaan tarvitaan omia

prosesseja ja työkaluja. Jos niitä ei ole, nimikemäärä voi kasvaa hallitsemattomasti, mikä aiheuttaa tehottomuutta, ylimääräistä työtä ja virheitä yrityksen toiminnassa. (Peltonen ym. 2002, 45.)

2.2.2 ERP-järjestelmät

ERP (Enterprise Resource Planning) eli toiminnanohjausjärjestelmä on tietojärjestelmä, jonka avulla organisaatio hallitsee tarkempia toimintojaan, esimerkiksi myynti, henkilöstöhallinto, taloushallinto sekä erilaisia toimialakohtaisia erityistoimintoja kuten muun muassa varastohallinta ja tuotannonohjaus. Näitä toimintoja hallitaan ERP-järjestelmän moduuleilla, joita voidaan lisätä järjestelmään yritystarpeiden mukaan. (Jokipii 2023.) ERP-järjestelmän mahdolliset moduulit on esitetty kuviossa 2.

ERP-järjestelmän päätehtävä on hallita, suunnitella, optimoida ja automatisoida yrityksen ydintoimintoja. ERP kytkee eri toiminnot toisiinsa ja sisältää suurimman osan organisaation tärkeimmistä tiedoista. (Jokipii 2023.)

Kuvio 2. ERP-järjestelmän moduulit (ite wiki)

ERP-järjestelmän avainhyödyt pätevät kaikille järjestelmän käyttäjille organisaation toimialasta riippumatta. Ensiksikin kaikki tärkeä tieto on saatavilla kaikille sidosryhmille yhdessä paikassa ajan tasaisesti. Toisekseen ERP-järjestelmä automatisoi yrityksen toiminnot ja vähentää manuaalisen työn sekä virheiden määrä ja parantaa työn tehokkuutta.

Kolmanneksi datan ajantasaisuus sekä oikeellisuus mahdollistavat dataan perustuvan päätöksenteon kaikkialla yrityksessä yksittäisestä operatiivisesta työntekijästä johtoportaan saakka. (Jokipii 2023.)

ERP-järjestelmien historia alkaa jo 1970-luvulta, jolloin yritykset aloittivat käyttämään tietokoneita toiminnassaan (Nestell ym. 2017, Tanskanen & Nivamo 2017 mukaan). Siitä ERP-järjestelmien kehittyminen lähti kasvuun ja 1990-luvulla ERP:n käyttö yleistyi sitä mukaa, kun isot IT-yritykset alkoivat tarjoamaan markkinoille valmiita ohjelmistoratkaisuja. Sitä ennen isommat yritykset tekivät omia ohjelmistojaan omalla työvoimalla. Nykyään ERP-järjestelmien toimittajia on lukematon määrä ja monenlaiset tietojärjestelmät ovat osa työntekijöiden arkea. (Tanskanen & Nivamo 2017.)

Kaikki organisaatiot, toimialasta ja koosta riippumatta, tarvitsevat ratkaisuja toiminnanohjaukseen, mutta esimerkiksi terveydenhuollon organisaation ja valmistavan teollisuuden yrityksen tarpeet eroavat toisistaan huomattavasti. Tänä päivänä ohjelmistotoimittajat tarjoavat omat ratkaisut esimerkiksi pienille ja keskisuurille sekä isoille yrityksille. Toimialakohtaiset tarpeet täytetään käyttämällä yrityksen toimintoja vastaavia ERP-järjestelmän moduuleja. ERP:n toiminnallisuutta voidaan laajentaa integroimalla sen muihin yrityksen tietojärjestelmiin. (Jokipii 2023.)

Perinteiset ERP-järjestelmät olivat organisaation omalle tai kolmannen osapuolen palvelimelle asennettuja itsenäisiä, muista tietojärjestelmistä irrallisia ohjelmistoratkaisuja. Perinteisen järjestelmän implementointi ja ylläpito on kallista, eikä mahdollista uusien teknikoiden käyttöä sekä prosessien optimointia. Tällaisten huonosti skaalautuvien järjestelmien elinkaari usein rajoittuu noin 5–10 vuoteen, minkä jälkeen organisaatiota odottaisivat laajat järjestelmäpäivitykset tai jopa uuden järjestelmän käyttöönottoprojekti. Nykypäivänä paikalliset ERP-ratkaisut ovat jäämässä historiaan, niiden tilalle tulevat pilvipohjaiset tietojärjestelmät. Pilvipohjainen ERP on joustava alusta, joka yhdistää yrityksen liiketoiminnan osa-alueet yhdeksi täysvaltaiseksi ratkaisuksi. Pilvipohjaisen järjestelmän käytön etuna voidaan pitää alustan skaalautuvuuden ansiosta sitä, että järjestelmistä on tullut laajempia, mutta joustavampia. Pilvijärjestelmäintegraation implementointi on helpompaa ja sen toimintavarmuus on ihan eri luokkaa yksittäiseen kahden järjestelmän väliseen integraatioon verrattuna. Lisäksi pilviteknologia auttaa organisaatioita pääsemään eroon järjestelmäpäivityksien asentamisesta, palvelinten ylläpidosta ja siihen liittyvistä kustannuksista, organisaatio maksaa vain järjestelmän käytöstä, eikä omistamisesta. Yleisen alustan ansiosta kaikki yrityksen tiedot ja tapahtumat ovat reaaliaikaiset ja kaikkien saatavilla ajasta, paikasta ja laitteesta riippumatta. (Jokipii 2023.)

Globaalin digitalisaation myötä datan merkitys yrityksen liiketoiminnassa kasvaa vuosi vuodelta. ERP-järjestelmiin kertyy paljon yrityksen moninaista dataa ja sen mukaan kasvaa ERP-järjestelmien merkitys yritykselle. (Tanskanen & Nivamo 2017.)

2.2.3 PDM/PLM-järjestelmät

1980-luvulla CAD-järjestelmät alettiin ottaa käyttöön valmistavassa teollisuudessa. Tämä mahdollisti tuotantoprosessin tuottavuuden ja tehokkuuden kasvun, mutta samalla syntyi tarve keskitetysti hallita tuotteita koskevia tietoja. Tähän tarpeeseen perustettiin tuotetietohallintajärjestelmät, eli PDM (Product Data Management) -järjestelmät. Seuraava vaihe näiden järjestelmien kehityksessä oli tuotetiedon sekä tuotekehitysvaiheiden hallinta tuotteen elinkaaren aikana. Tässä tapauksessa puhutaan tuotteiden elinkaaren hallinnasta, eli PLM:stä (Product Lifecycle Management). Ensimmäiset PLM-järjestelmät kehitettiin vuosituhannen vaihteessa. (Trebuna ym. 2013.)

2000-luvulla PDM/PLM-järjestelmät ovat kehittynet valtavasti. Monien muiden yritysten tietojärjestelmien tapaan ne integroidaan muihin järjestelmiin ja yhä useammin pilvipohjaiset ratkaisut korvaavat perinteiset työpöytäsovellukset. Kaikki edellä mainitut pilvipohjaisten ratkaisujen edut, kuten ylläpitotoimintojen ja kustannusten vähentyminen, alustan skaalautuvuus sekä datan ajantasaisuus ja saatavuus ajasta, paikasta ja laitteesta riippumatta pätevät hyvin myös uusiin PDM/PLM-järjestelmiin.

Nykypäivänä tuotteet ja tuoterakenteet monimutkaistuvat, tämä kasvattaa tuotetiedonhallinnan vaatimuksia. Tuotteiden suunnittelu tarvitsee hyvän muutos- ja versiohallinnan, lisäksi tuoteversioiden julkaisun pitää olla hallittua ja niihin liittyvän dokumentaation tulee olla saatavilla kaikille sitä tietoa tarvitseville sidosryhmille. (Tiuttu 2019.)

Tuotetiedon ytimenä ovat nimikkeet ja nimikerakenteet, sekä tuotekehitykseen liittyvä dokumentaatio. Suuri osa nimikkeistä perustetaan tuotekehityksen ja suunnittelun tarpeisiin, siksi PDM/PLM-järjestelmä yleensä on nimikemasterin roolissa. (Tiuttu 2019.) Kyseisen tiedon masteriksi kutsutaan järjestelmää, jossa tietoja hallitaan ja muokataan (Peltonen ym. 2002, 107).

Tuotekehityksen ja projektisuunnittelun data koostuu pitkälti CAD-ohjelmien tuotoksista, kuten 3D-mallit, piirustukset, mekaniikka- ja sähködokumentaatio. CAD-ohjelmien integraatio PDM/PLM-järjestelmään auttaa saamaan koko tuotteen tiedon yhdeksi tuoterakennenäkymäksi ja siksi integraatio on välttämätöntä yrityksen liiketoiminnan tehokkuuden kannalta. Nimikkeisiin liittyvät dokumentit tallennetaan PLM/PDM-järjestelmään, josta niitä haetaan ja tarvittaessa tuotetietoa jaetaan asiakkaille, toimittajille sekä alihankkijoille. Tuotedokumenttien siirtoa muihin järjestelmien ei suositella. (Tiuttu 2019).

2.2.4 PDM/PLM- ja ERP-järjestelmien työnjako

ERP-järjestelmien avulla hallitaan lähes kaikkia yrityksen tietoja ja toimintoja, siksi tuotetiedonhallinta voidaan ajatella kuuluvaan ERP-järjestelmille. Kuitenkin ERP on etupäässä prosessiorientoitunut järjestelmä. Sen sijaan PDM-järjestelmät nimiensä mukaan käsittelevät tuotteita ja niihin liittyvää dataa. Yritykset tarvitsevat sekä ERP- että PDM-järjestelmän integroituna toisiinsa. Molemmissa järjestelmissä käsitellään osittain samaa tietoa, joten työnjako näiden järjestelmien välissä on erityisen tärkeä yrityksen tietojärjestelmän suunnittelussa. (Peltonen ym. 2002, 11; Tanskanen & Nivamo 2017.) Yleisesti voidaan määritellä, että PDM/PLM on tuotetiedon masteri ja ERP on tuotantoon sekä transaktioihin liittyvän tiedon masteri (Tiuttu 2019). Nimikkeet ja nimikerakenteet yleensä luodaan PDM/PLM-järjestelmään, siinä myös muokataan nimikkeiden perus- ja tekniset tiedot (esimerkiksi tuotekuvaus, materiaali). Integraation avulla nimikkeet siirrettään ERP-järjestelmään, jossa nimiketietoa rikastetaan yrityksen liiketoimintaprosesseihin liittyvillä attribuuteilla (esimerkiksi toimittaja, tullikoodi). Alla on lueteltu järjestelmien keskeiset toiminnot. Kuvio 3 auttaa hahmottamaan järjestelmien työnjakoa.

PDM/PLM-järjestelmässä hallitaan:

- tuotteet
- tuoterakenteet
- dokumentaatio
- tuotemuutosprosessit
- huolto- ja varaosatieto.

ERP-järjestelmässä hallitaan:

- myynti
- tilaukset
- hankinnat
- tuotantoprosessit
- varastot
- toimitukset. (Tiuttu 2019.)

Kuvio 3. PDM/PLM ja ERP - tehtävät ja roolit (Tiuttu 2019)

Järjestelmien roolit muodostavat järjestelmien väliset integraatiolinkit ja määrittelevät, mikä tieto siirtyy integraation kautta. Tärkeimmät ovat nimike- ja nimikerakennesiirrot PDM/PLM:stä ERP:iin. Tarve siirtää tietoja ERP:stä PDM/PLM:ään ei ole välttämätön ja riippuu yrityksen toiminnasta. ERP:stä PDM:ään voidaan siirtää nimikkeiden ei-tekniset attribuutit kuten esimerkiksi asiakastiedot. (Tiuttu 2019.)

Tiutun (2019) mukaan PDM/PLM-järjestelmät edustavat yrityksen teknologista tuotenäkökulmaa, ERP-järjestelmät operatiivista myynnin ja tuotannon näkökulmaa. Järjestelmien välinen työnjako riippuu organisaation toimintamallista, mutta kummatkin järjestelmät ovat liiketoiminnalle elintärkeitä. Tästä syystä tietojärjestelmien tulisi toimia tehokkaasti saumattomassa yhteistyössä.

2.3 Yrityksen datan laatu ja nimikehallinnan keskeiset haasteet

Tässä luvussa datalla tarkoitetaan etupäässä yrityksen master dataa. Nimiketieto on tärkeä osa yrityksen master dataa ja kaikki alla mainitut datan laatuun liittyvät asiat ovat vahvasti kytköksessä nimikedataan ja sen hallintaan.

2.3.1 Hyvä datan laatu

Kirjassaan Master data Väre (2019, 199–200) kertoo, että master datan hallinnan pääperiaatteet ovat datan tunnistettavuus sekä saatavuus ja monipuolinen hyödynnettävyys läpi organisaation. Master data pidetään hyvänlaatuisena silloin, kun siitä voi luotettavasti yksilöidä todellisen elämän tosiasiaa kuten esimerkkisi henkilö, tuote tai mikä tahansa muu tietue ja se sisältää kaikki eri sidosryhmien tarvittavat attribuutit. Datan saatavuuden kannalta hyvä master datan laatu merkitse sitä, että dataan tehtävät päivitykset jaettu organisaatiossa kaikkialle, missä kyseistä data käytetään, toisin sanoen hyvän datan täytyy olla

tasaista ja ajankohtaista. Kaikkien datan attribuuttien on oltava sisällöllisesti ja muodollisesti tarkoitukseen sopivia. Järjestelemissä ylläpidettävien tietojen sopivuuden määrittelee datan käyttäjä. Datan käyttäjät yleensä ovat yrityksen liiketoiminnot ja asiakkaat. Siksi datan laadun ymmärtämiseen ja määrittelyyn tarvitaan yrityksen eri liiketoimintojen edustajat, joilla on käsitys siitä, miten asiakkaat käyttävät dataa.

Laadukas data edistää organisaation toimintojen monia puolia. Virheellinen ja puutteellinen data on este sujuvalle liiketoiminnalle, hyvänlatuinen data sen sijaan säästä työaikaa, resursseja ja parantaa organisaation tehokkuutta sekä pienentää kustannuksia. Datan laatuvirheet vaikuttavat organisaation maineeseen, sillä nykypäivänä ne tulevat herkemmin esille digitaalisten palvelukanavien kautta ja saattavat heikentää asiakkaiden brändimielikuvaa. Laajamittaisista datavirheistä (esimerkiksi virheellisesti lähetetyistä laskuista) on syntynyt isojakin haittoja yrityksen maineelle. Luotettavasta datasta organisaatiossa voidaan muodostaa laadukkaampaa informaatiota. Luotettavan informaation perustella voidaan vuorostaan tehdä luotettavampia päätöksiä. Hyvin varmistettu datan oikeellisuus ja ainutlaatuisuus estävät epäselvyyksiä, jotka mahdollistaisivat petoksia ja taloudellisia menetyksiä luottotappioina. Lisäksi hyvän datan avulla organisaatiot tavoittelevat parempaa asiakastyytyväisyyttä, ympäristöystävällisyyttä ja säädösten mukaisuutta. Varmistamalla datan laatua yritykset voivat saada paljonkin etuja nykyisessä digitaalisessa maailmassa. (Väre 2019, 201–202.)

2.3.2 Datan laadun hallinta

Hyvä datan laatu ei tule itsestään. Laatuvirheistä ei koskaan pystytä välttymään täysin, mutta yrityksen data ei voi olla hyvänlaatuista, jos siihen ei kiinnitetä erityistä huomiota. Väreen (2019, 191–192) mukaan datan laadun hallinta on toimintamalli, joka koostuu prosesseista, joilla yrityksessä ohjataan, seurataan, mitataan ja parannetaan datan laatua. Datan hyvä laatu on tavoite, johon siitä vastaavat henkilöt pyrkivät toiminnallaan. Datan laadun merkityksen tietoisuuden kasvattaminen organisaatiossa on tärkeä osa tätä toimintamallia. Datan omistajat vastaavat datan laadusta. He määrittelevät datan laatuvaatimukset ja laatua koskevat säännöt Datan laadun parantaminen ei kannata, ennen kuin tiedetään, mitä tarkoittaa hyvälaatuinen data. Kun työstetään laatusääntöjä ja vaatimuksia datan laadunvalvonnan alaisuuteen valitaan liiketoiminnalle kriittistä dataa. Siksi datan laadunhallinnan toimenpiteet kohdistetaan ensisijaisesti yrityksen master dataan.

Datan laadunhallinnan tarkoituksena on olla jatkuva ja kehittyvä prosessi, ei pelkkä tekninen toimenpide eikä kertaluontoinen datan siivousoperaatio. Datan ladunhallinta on osa koko organisaation datahallintaa. Sen lähtökohtana on liiketoiminnan tavoitteet ja

asiakkaiden tarpeet. Datan laadunhallinnalla on kuitenkin eri painopisteet, jotka ovat esitetty taulukossa 2. (Väre 2019, 193.)

REAKTIIVINEN → PROAKTIIVINEN

ONGELMAT	DATA	PROSESSIT	ASIAKASKOKEMUS
Raportoidaan yksittäisiä ongelmia	Raportoidaan datan trendejä	Raportoidaan tehokkuutta	Raportoidaan liiketoiminnan tavoitteisiin pääsyä
Datan laatu määritellään tapauskohtaisesti	Sisäiset datan käyttäjät määrittelevät datan laadun	Liiketoimintaprosessit määrittelivät datan laadun	Datan laatu määritellään tavoitteiden ja asiakaskokemuksen kautta
Manuaalisia ja kertaluontoisia toimenpiteitä	Jatkuvia toimenpiteitä	Datan hallinnan prosessit implementoitu, ettei samat laatu ongelmat ilmaannu uudelleen	Datan laatu huomioidaan jo suunnitteluvaiheessa prosesseihin ja sovelluksiin

SISÄÄNPÄINKATSOVA → ULOSSUUNTAUTUNUT

Taulukko 2. Datan laadunhallinnan painopisteet (Väre 2019, 193)

Väreen (2019, 194–195) mukaan reaktiivisten toimenpiteiden painopiste on pelkässä datassa ja siihen liittyvissä ongelmissa. Virheet korjataan vasta siinä vaiheessa, kun huonosta datasta on muodostunut lisäkustannuksia yritykselle. Proaktiivisen laadunhallinnan painopisteet ovat liiketoiminnan prosesseissa ja asiakaskokemuksessa. Proaktiivinen toimintatapa vähentää datan laatuvirheistä aiheutuneita kustannuksia merkittävästi. Parhaimmillaan huolehtiminen hyvästä datan laadusta kuuluu organisaation päivittäisiin toimintatapoihin, kun datan laatu huomioidaan datan luomisessa sekä muokkaamisessa (”data quality by design”). Silloin datan laatu on vahvasti osana yrityksen liiketoimintaprosesseja. Myös järjestelmien käyttöliittymien ja integraatioiden suunnittelussa otetaan huomioon datan laatu rakentamalla järjestelmät, joiden avulla virheellisen datan syntymistä voisi minimoida. Datan määrä kasvaa jatkuvasti, joka kasvattaa myös datan laadun parantamisen toimenpiteiden määrää. Yrityksen datan laatuvirheiden kustannukset kasvavat ajan mukaan, sillä mitä myöhemmin virheet havaitaan ja korjataan, siitä kalliimmaksi ne tulevat. Laatuvirheiden syntymisen estäminen tai havaitseminen varhaisessa vaiheessa on kustannustehokkaampaa organisaatiolle. Taulukossa 3 on esimerkki datavirheiden vaikutuksesta tuotteen elinkaaren eri vaiheissa.

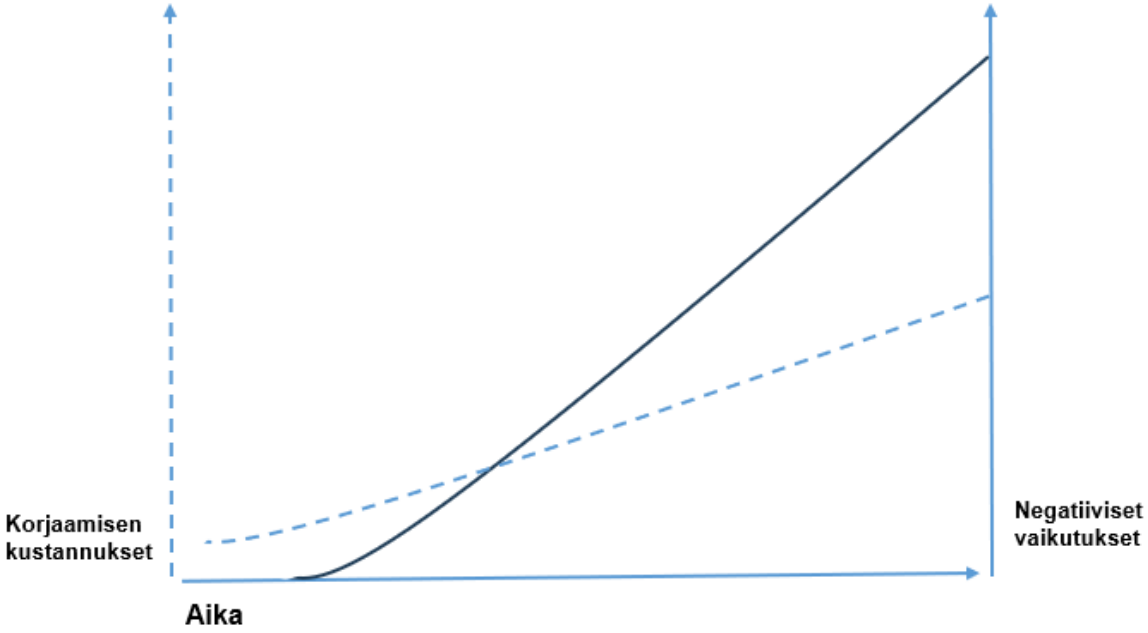
Tilanne: Uuden miesten vaatemerkin tuotteita perustetaan ERP-järjestelmään lasten tuotekategoriaan.

Tuotekategorialla ohjataan myyntitavoitteita, hankintaa, tuotteiden näkymistä verkkokaupassa, varastointia ja toimitusta oikealle osastolle myymälöissä. Kategorian numerokoodi on merkitty tuotteen hintalappuun, muutoin se ei näy tuotteesta ulospäin.

Vaihe	Vaikutus	Korjaamisen kustannukset
Tuote perustetaan ja tilataan toimittajalta	Ei negatiivisia vaikutuksia	Tuotekategorian korjaamiseen ERP:ssa tarvittava työaika
Toimittaja lähettää tilauksen varastoon	Virheellisen tullitariffin vuoksi tullausvaiheessa voi syntyä ylimääräisiä kustannuksia, viivästyksiä tai jopa sanktioita tullilta	Tullausasiakirjojen korjaamiseen tarvittava työaika
Varasto hinnoittelee ja lähettää tuotteet myymälöihin	Tuotteet toimitetaan väärälle osastolle, hyllytys viivästyy	Tuotteiden uudelleen hinnoitteluun tarvittava työaika, hintalappujen uusimisen aiheuttaa lisäkustannukset
Tuotteita myydään	Tuotteiden väärä ryhmittely verkkokupassa aiheuttaa asiakkaiden tyytymättömyyttä sekä menetettyä myyntiä, koska tuotteita on vaikea löytää	Tuotekategorian korjaamiseen verkkokaupassa tarvittava työaika; tietovaraston myyntihistorian konvertointi vaatii työaikaa sekä teknisiä kustannuksia
Edellisen kauden myynnin raportointi ja uuden kauden budjetointi	Tuotekategorioittain otettavien myyntiraporttien vääristyminen aiheuttaa budjetoinnin vääristymisen, seuraavalla kaudella saatetaan hankkia liikaa lastenvaatteita ja liian vähän miesten vaateita	Raporttien ja budjettien korjaamiseen tarvittava työaika

Taulukko 3. Virheellisen datan vaikutus yrityksen toimintaan (Väre 2019, 196)

Taulukosta on luettavissa, miten yrityksen kustannukset kasvavat virheellisen datan etenemisen mukaan. Jokaisen seuraavan vaiheen kustannukset tulee lisätä edellisten vaiheiden kustannuksiin, toisin sanoen virheen vaikutukset ja kustannukset kumuloituvat virheen elinajan aikana. Heti alussa virheiden korjaamisen kustannukset olisivat alhaisimmat, mutta siinä vaiheessa virheen havaitseminen on vaikeampaa, koska tuote on olemassa vain virtuaalisesti järjestelmässä. Tullausvaiheessa virheiden kustannukset olisivat kohtuullisen alhaisia, mutta virheelliset tullitariffit usein jäävät huomaamatta tullissa. Varastoinnin yhteydessä virheellinen tuotekategoria voitaisiin havaita, jos työntekijät tuntevat tuotekategoriat hyvin. Silloinkin virhe olisi voitu korjata suhteellisen pienellä vaivalla ja kustannuksilla. Kuitenkin todennäköisemmin virhe havaitaan vasta verkkomyynnissä, jolloin väärä kategoriaryhmittely tulee asiakkaille näkyviin. Tällöin virheiden negatiiviset vaikutukset kasvavat isosti, väärin kategorioidut tuotteet vaikeuttavat hakua verkkokaupassa, joka puolestaan heikentää yrityksen mainetta sekä laskee tuotteiden myyntiä. Myyntihistorian korjaaminen tietovarastossa myös aiheuttaa suuria kustannuksia. Pahimmassa tapauksessa virhettä ei korjata ollenkaan ja seuraavan kauden myyntitavoitteet suunnitellaan menneen kauden pohjalla. Tällöin virheellinen data voi jopa laskea yrityksen liikevaihtoa ja pudottaa myyntikatetta. Ajan kuluessa negatiiviset vaikutukset kasvavat huomattavasti suuremmiksi kuin virheiden korjaamisen kustannukset. Kuitenkaan negatiiviset vaikutukset eivät ala heti, kuten näkyy kuviosta 4. Virheiden korjaamisen kustannukset sen sijaan alkavat muodostumaan melkein heti, kun virhe on tehty. Ainoa tapa välttyä virheiden korjaamisen aiheuttamista kustannuksista on olla tekemättä virheitä, mikä on käytännössä mahdotonta. Mutta kuten kuviosta näkyy, korjauskustannukset pysyvät niin kauan suhteellisen matalina, kunnes negatiiviset vaikutukset alkavat kasvaa. (Väre 2019, 197–198.) Joten yritysten on määriteltävä, miten usein datan laatua pitäisi huoltaa ja mihin liiketoiminnan prosesseihin huoltotoimenpiteet pitäisi kohdistaa.



Kuvio 4. Datan laatuvirheiden vaikutukset (Väre 2019, 198)

2.3.3 Datan laadun ulottuvuudet

Laadukasta dataa on data, joka soveltuu tiettyyn tarkoitukseen. Tämä määritelmä viittaa siihen, että data soveltuu liiketoiminnan tavoitteiden saavuttamiseen sekä tehokkaiden päätösten tekemiseen. Vaikka tämä yksinkertainen määritelmä on hyvä lähtökohta, se ei kerro, miten tietojen laadun parantamiseen tulisi suhtautua yksittäisissä käyttötapauksissa. (Varshney 2023.) Datan laatuvirheitä tutkimalla huomaa, että virheissä sekä niiden vaikutuksessa liiketoimintaan on eroja. Virheiden luokittelussa voi auttaa datan laadun käsittely eri ulottuvuuksista. Silloin kun ymmärtää virheiden syitä ja luonnetta, oppii paremmin näkemään virheiden korjaamisen tapoja. (Väre 2019, 203.) Kuvio 5 esittää yleisimmät datan laadun ulottuvuudet, jotka on hyvä ottaa huomioon datan laadun parantamisessa.

Kuvio 5. Datan laadun ulottuvuudet (Varshney 2023)

Completeness eli sisällön kattavuus kertoo, onko dataa olemassa. Puuttuva data on suuri ongelma yrityksen liiketoiminnalle. Jos ei ole varmuutta datan kattavuudesta, kaikki datan perusteella tehdyt analyysit, raportit tai päätökset ovat todennäköisesti virheellisiä. Esimerkiksi markkinointisuunnitelmaa laatiessa on tärkeää tietää yrityksen kohdeasiakkaiden tiedot. Jos tärkeitä tietoja, kuten ikä, sukupuoli, työllisyystilanne tai asuinpaikka puuttuu, on mahdotonta luoda tarkkaa kuvaa asiakaskunnasta. On aina varmistettava, että data sisältää kaikki tärkeät tiedot, tai ainakin pitää olla käsitys siitä, mikä data puutuu, että voisi ottaa sen huomioon. (Varshney 2023; Väre 2019, 204.) Datan olemassaolo on suhteellisen helppoa tarkistaa. Ensin etsitään, onko tietojärjestelmässä paikkaa eli kenttää, jossa dataa ylläpidetään ja sitten tarkistetaan, onko se tyhjä vai ei. On hyvä muistaa, että liiketoiminnan kannalta tyhjä kenttä voi olla täytetty teknisessä mielessä, toisin sanoen se voi sisältää esimerkiksi nollia tai välilyöntejä. Tarkistuksessa kannattaa huomioida sekä tyhjiä niin sanottuja ”null” arvoja, että nollat ja välilyönnit sisältävät kentät. Datan sisällön kattavuutta voidaan arvioida joko attribuuttitasolla (yhden attribuuttikentän tarkastus sarakkeittain) tai tietuekohtaisesti (tarkistetaan yhden rivin kaikki attribuutit). Tietuetason kattavuus yleensä antaa paremman kuvan tilanteesta kuin yksittäisten attribuuttien tarkastelu. Datan sisällön kattavuus on erityisen tärkeä ominaisuus muiden datan ulottuvuuksien mittaamisessa, varsinkin ainutlaatuisuuden arvioinnissa. (Väre 2019, 204–205.)

Accuracy eli oikeellisuus kertoo, vastaako olemassa oleva data tosielämää. Tämä on tärkein datan laadun ulottuvuus. Monien alojen (esimerkiksi terveys- tai finanssiala) toimintojen kannalta datan oikeellisuus on kriittistä. Jos data on sisällöltään kattava, sen oikeellisuus täytyy arvioida. Esimerkiksi osoitekentässä voi olla teksti ”ei tiedossa”, tai kentät voivat olla täytetty oletusarvoilla. Näistä tiedoista ei ole hyötyä kenellekään. (Väre 2019, 206.) On kaksi käytännön tapaa arvioida datan oikeellisuutta. Datan profilointi (data profiling) on prosessi, jossa data analysoidaan epäjohdonmukaisuuksien, virheiden ja muiden poikkeamien tunnistamiseksi. Datan validointi (data validation) varmistaa datan oikeellisuutta ennalta määritettyjen kriteerien perusteella, esimerkiksi vertailemalla olemassa olevat arvot luotettavan lähteen toimittamiin standardi- tai viitearvoihin. (Varshney 2023.)

Uniqueness eli ainutlaatuisuus kertoo, että data esiintyy järjestelmässä tai tietokannassa vain kerran. Datan ainutlaatuisuuden huomioiminen on todella tärkeää, sillä päällekkäiset tiedot voivat aiheuttaa epäselvyyksiä raportoinnissa ja analyyseissä tai johtaa harhaan päätöksentekoprosessia. Jos esimerkiksi myyntitapahtuma on kirjattu järjestelmään kahdesti, näyttää siltä, että yritys on saanut enemmän tuloja kuin todellisuudessa, mikä vääristää tuloslaskelmat ja ennusteet. Sisällön kattavuus ja oikeellisuus helpottavat huomattavasti ainutlaatuisuuden varmistamista. Puuteellisen datan ainutlaatuisuuden tarkistaminen voi olla haastava. Yrityksellä on oltava säännöt, joilla arvioidaan datan ainutlaatuisuus ja käsitellään poikkeustapaukset ja puutteet datassa. (Varshney 2023; Väre 2019, 208.)

Validity eli vaatimustenmukaisuus viittaa siihen, että data vastaa sille määriteltyjä muodollisia ja sisällöllisiä vaatimuksia. Muodollinen kelpoisuus edistää datan tulkittavuutta järjestelmässä ja järjestelmien välistä siirrettävyyttä. Esimerkiksi sähköpostiosoitteen pitää sisältää @-merkki, eikä siinä saa olla välilyöntejä. Jos tätä muotoa ei noudateta, automatisoitu viestien lähettäminen ei onnistu. Sisällöllisten vaatimusten täyttyminen tarkoittaa, että data on kelvollinen liiketoiminnan vaatimusten kannalta. Esimerkiksi, jos yrityksen henkilöstötiedoissa pitää olla koko nimi, pelkkä sukunimi ei riitä. (Väre 2019, 209–210.) Yrityksellä täytyy olla selkeät säännöstöt, joiden perusteella luodaan uutta dataa. Järjestelmien käyttöliittymät on myös usein suunniteltu niin, että muodollisesti virheellisten tietojen (esimerkiksi päivämäärät, mittayksiköt) syöttäminen ei ole mahdollista.

Consistency eli yhteneväisyys Väreen kirjan (2019, 210–211) mukaan tarkoittaa, että datan on oltava sama, riippumatta siitä, missä järjestelmässä tai tietokannassa se esiintyy. Datan päivityksen yhteydessä on huolehdittava siitä, että tiedot on päivitetty kaikkialla, missä kyseistä data käytetään. Puutteet datan yhteneväisyydessä tuottavat harmia sekä yrityksen sisällä, että ulkopuolella. Ne voivat aiheuttaa asiakaspalveluongelmia tai jopa mainehaittoja, jos esimerkiksi asiakkaita lähestytään väärillä tiedoilla. Yhteneväisyyttä tarkistetaan

selvittämällä, mistä järjestelmistä ja miten kyseisen datan voi löytää, jonka jälkeen eri järjestelmien tietoja vertaillaan keskenään. Haasteena tässä operaatiossa on järjestelmien erilaisuus. Usein eri järjestelmissä data tallennetaan eri muodossa, mikä tekee suoran vertailun mahdottomaksi. Esimerkiksi toisessa järjestelmässä asiakkaan etu- ja sukunimi voi olla samassa kentässä, toisessa on omat kentät kummallekin. Päivämääräformaattien ja tuotekoodien muodot myös voi erota eri järjestelmissä. Siinä tapauksessa data täytyy standardoida eli yhdenmukaistaa ennen tarkistusta. Tämä vaihe saattaa olla työläs, mutta on toteutettavissa oikeilla teknisillä työkaluilla.

Integrity eli eheys Väreen (2019, 212–213) mukaan osoittaa, että järjestelmien väliset viittaukset ovat oikein ja tietty master datan tietue voidaan yhdistää luotettavasti eri järjestelmien välissä. Viitetietoja on järjestelmissä käytettävät avaintiedot, joiden avulla voidaan yhdistää dataa järjestelmien tai tietokantojen kesken. Avaimena usein käytetään tietuen yksilöivää tunnistetta, kuten henkilötunnusta tai tuotekoodia. Jos viiteavaimet eivät kohdistu oikein tietokannan sisällä tai järjestelmien välissä, dataa ei voi yhdistää tai se yhdistyy väärin. Tämä voi aiheuttaa virheitä ja puutteita datassa. Datan oikeellisuus on tärkeä eheyden tarkistuksessa. Eheys varmistaa, että viitteet toimivat, oikeellisuudella varmistetaan, että viittaus kohdistuu oikein. Jos järjestelmässä viitataan toisessa järjestelmässä olevaan asiakkaan, on varmistettava, että kyseessä on sama asiakas.

Timeliness eli ajanmukaisuus kerto, onko data oikea-aikaista ja on saatavilla oikeaan aikaan. Jos datan oikeellisuudesta huolehditaan, master datan kannalta ajanmukaisuus usein ei ole niin tärkeä ominaisuus. Siinä on poikkeuksia, esimerkiksi terveysalalla, jossa potilaiden tietojen on oltava viimeisintä parhaan hoidon saamiseksi. Ajanmukaisuutta voi käyttää apuna datan oikeellisuuden varmistamisessa. Jos dataa ei ole päivitetty vuosiin, sen ajanmukaisuutta ja oikeellisuutta olisi hyvää tarkistella. Vertailemalla datan päivityksen ajankohtia eri järjestelmissä, voidaan havaita, jos joku järjestelmä ei päivity oikein. (Väre 2019, 214.)

Dataa voidaan käsitellä eri ulottuvuuksien kannalta. Väre (2019, 214) toteaa, että kaikki ulottuvuudet eivät ole yhtä tärkeitä yrityksen master datalle. Kaikkia ulottuvuuksia ei välttämättä tarvitse seurata kaiken datan osalta, vaan pitää valita tärkeimmät ominaisuudet jokaiselle attribuutille. Datan laadun parantamisessa kannattaa edetä vaiheittain siirtyen ulottuvuuksista toiseen. Esimerkiksi datan oikeellisuutta voidaan arvioida vasta, kun datan sisältö on kattavuudeltaan saatu kuntoon.

2.3.4 Virheiden syyt ja datan laadun parantaminen

Kuvio 6 esittää master datan 80/20 hallintamallin, josta selviää, että datahallinnan prosesseissa 80 % työpanosta ja vastuuta on yrityksen liiketoiminnoilla eli datan käyttäjillä ja vain 20 % kuuluu teknologioille ja IT-järjestelmille.

Ihmiset ja toimintatavat • Master datan hallintamalli • Visio, strategia ja tavoitteet • Master datan määrittely ja standardointi • Master datan hallinnan prosessit ja elinkaari • Master datan laadun mittaaminen ja seuranta	80 % työpanoksesta
Arkkitehtuuri ja teknologiat	20 % työpanoksesta

Kuvio 6. Master datan hallinnan 80/20 malli (Väre 2019, 64)

Lisäksi valtaosa yritysten järjestelmien ylläpitämästä datasta on edelleen ihmisten syöttämää, joten on ihan luonnollista, että suurin osa datan laatuvirheistä johtuu ihmisten toiminnasta, eikä järjestelmistä. Väreen (2019, 219) mukaan yleisimmät inhimillisten virheiden syyt ovat muun muassa kiire, ristiriitaiset ohjeet tai ohjeiden puute sekä motivaation puute.

Joskus on kiire saada asiat valmiiksi nopeasti. Silloin huolimattomuudessa voidaan syöttää virheellistä dataa järjestelmään esimerkiksi, kun tehdään näppäilyvirheitä tai epävarmoissa tilanteissa ei ehditä tutustua ohjeisiin. Kiireen haittavaikutukset moninkertaistuvat, kun datan laadun hallintaprosesseihin ei panosteta riittävästi tiukkojen aikataulujen vuoksi. (Väre 2019, 225–226.)

Väre (2019, 226–227) toteaa, että suuri osa laatuvirheistä johtuu siitä, että organisaatiolla ei ole datan käsittelyn liittyviä sääntöjä tai ne ovat puutteelliset tai vaikeasti saatavilla. Isoissa organisaatioissa datakäyttäjiä ja sidosryhmiä on paljon, joten kaikista prosesseista, jotka liittyvät datan luomiseen, ylläpitämiseen, päivittämiseen ja poistamiseen on oltava selkeät yksiselitteiset ohjeet, joissa kerrotaan, mitä dataa syötetään mihin kenttiin ja miksi. Ohjeiden pitää olla hyvin löydettävissä aina, kun niitä tarvitaan. Ohjeiden puutteellisuus tai puute usein vaikuttavat ihmisten työmotivaatioon. Jos ei tiedetä syytä, miksi asiat tehdään näin, ei ole motivaatiota tehdä asioita oikein, vaan usein yritetään päästä helpommalla. Myös asiakkaiden motivaation puute on yleinen ilmiö, esimerkiksi yritysten laatimiin monenlaisiin pakollisiin kyselyihin vastaamisessa. Joten, jos halutaan saada tietoa asiakkaalta, on mietittävä tarkkaan, miten kannattaa lähestyä häntä.

Jossain tilanteissa käyttäjän syöttämän tiedon virheellisyys johtuu tietojärjestelmän käyttöliittymän epäselvyydestä. Silloin käyttäjällä ei ole varmuutta, mitkä arvot pitää syöttää. Tässä tapauksessa virheiden syynä voi pitää huonosti suunniteltua käyttöliittymää. Esimerkiksi jonkun kentän pakollinen täyttö voi johtaa siihen, että käyttäjän on pakko syöttää täysin virheellistä tietoa tai oletusarvoa, jos oikea tieto tuotteesta ei ole saatavilla tuotteen avaamiseen järjestelmään yhteydessä. (Väre 2019, 227–228.)

Väreen kirjassa (2019, 220) kerrotaan, että datan laatua voidaan parantaa monin eri tavoin. Laadun parantamisen toimenpiteitä voi tehdä kertaluontoisena projektina. Yleensä on tapana siivota dataa ainakin uuden järjestelmän käyttöönoton tai datan migraation yhteydessä. Yhteensopivuuden syistä nämä siivousmenetelmät usein ovat välttämättömiä. Kuitenkaan jos datan laatua ei huolleta jatkuvasti eikä laatuvirheiden juurisyihin puututa, datan laatu palaa aiemmalle tasolle pian projektin päättymisen jälkeen. Yritykset tarvitsevat toimintamallin saadakseen pysyvät muutokset datan laatuun.

Kuten monissa muissa kehittämisprosesseissa datan laadun parantamisessa voidaan käyttää PDSA kehityssykliä (kuvio 7). PDSA-sykli (Plan-Do-Study-Act) on oppimisen ja tiedon hankkimisen järjestelmällinen prosessi tuotteen, menetelmän tai palvelun jatkuvaa parantamista varten. Tämä integroitu oppimis- ja kehitysmalli tunnetaan myös nimellä Demingin laatuympyrä. (Deming Institute 2023.)

Kuvio 7. Demingin laatuympyrä (Deming Institute 2023)

Sykli alkaa suunnitteluvaiheella (Plan), joka sisältää tavoitteen asettamisen, sopivien menetelmien ja mittareiden vallinnan (Deming Institute 2023). Tässä vaiheessa mietitään, mikä olisi ensimmäinen askel datan laadun parantamisessa. Se voi olla kriittisin eniten lisäkustannuksia aiheuttava ongelma tai päinvastoin helpommin ratkaistava haaste, jolla voi osoittaa, että valitut menetelmät toimivat ja työ tuottaa tuloksia. Kun ensi askel on päätetty, suunnitellaan, miten ongelma ratkaistaan. Korjaustoimenpiteet riippuvat ongelmien juurisyistä, joten on hyvä tutkia, mikä tekijä aiheuttaa kyseistä ongelmaa ja missä vaiheessa virhe syntyy. Kuten aiemmin kerrottu, laatuvirhe aiheuttaa kustannuksia melkein heti, kun se on tehty. Mutta virheen negatiivisia vaikutuksia voidaan vähentää huomattavasti, jos virhe korjataan varhaisessa vaiheessa. Jos virheen syntymiseltä ei voi välttyä täysin, kannattaa miettiä, millä laadunvarmistuksen toimenpiteillä voisi pienentää laatuvirheiden lisäkustannukset sekä negatiiviset vaikutukset. Valittujen toimenpiteiden onnistumista voidaan mitata eri tavoin joko määrällisesti virheellisten ilmentymien lukuina tai prosenttiosuutena tai vaikka laadullisesti haastattelemalla datan käyttäjiä. Mittauksen vertailuarvona voi olla esimerkiksi tavoitearvo, alin hyväksytty arvo tai nykytilanne. Hyvä mittari ei kerro pelkästään arvosta, vaan myös siitä, miksi mitattava luku on tärkeä. Ladun valvontaan ja raportointiin kannattaa valita kipeimpiä ensisijaisesti ratkaisua vaativia ongelmia. (Väre 2019, 216–217, 222.)

Kun tavoite on asetettu ja ratkaisumenetelmät päätetty, siirrytään toimintavaiheeseen (Do). Väreen (2019, 223) mukaan siinä vaiheessa toteutetaan datan laadun parantamissuunnitelma. Suunniteltu toimenpide voi olla prosessin tai käyttöliittymän muutos, ohjeistus tai koulutus, joka voi olla kohdistettu isommille käyttäjäryhmille tai tietyille prosessin suorittajille, jos virheen juurisyy johtuu käyttäjien toimintatavoista. Tärkeitä on toteuttaa vain yksi muutos kerrallaan, sillä toimenpiteen vaikutusta voisi mitata luotettavasti. Menetelmät kannattaa toteuttaa testauksen tai soveltuvuusselvityksen (proof-of-concept) kautta.

Toteuttamisvaiheen jälkeen tarkistetaan suunnitelman toimivuutta (Study). Jos testauksen ja mittaamisen tulokset vastaavat odotuksia, kyseinen toimenpide suoritetaan loppuun viemällä sen koko organisaation tasolle ja varmistetaan sen positiivinen vaikutus ennen uusien toimenpiteiden aloittamista. Jos testauksen tulokset eivät tuota odotettua tulosta, on palattava edellisiin vaiheisiin ja tutkittava asiaa uudelleen. Oliko valittu toimenpide oikea kyseisen ongelman ratkaisemiseen vai tuliko toteutusvaiheessa teknisiä esteitä tai uutta tietoa ongelmasta? Esimerkiksi käyttöliittymän korjaus voi olla liian kallis implementoida, jolloin datan laatua voisi parantaa ohjeistamalla tai kouluttamalla datan käsittelijöitä. (Väre 2019, 224.)

Seuraava vaihe (Act) sulkee ympyrän kokoamalla yhteen koko prosessin tuottaman oppimisen, jonka avulla voidaan muuttaa tavoitetta tai menetelmiä, muotoilla tietoja ongelmasta

tai laajentaa oppimis-parannussykliä pienimuotoisesta kokeilusta isompaan toteutussuunnitelmaan. Tämän vaiheen tärkein anti on prosessin tuloksista ja havainnoista saatu tieto. Jos prosessi onnistui hyvin, on varmistettava, että toimenpiteet on otettu käyttöön koko organisaatiossa ja kaikki vaiheet ovat huolellisesti dokumentoitu ja uudelleen käytettävissä samankaltaisten ongelmien ratkaisemisessa. Jos prosessi ei tuottanut odotettuja tuloksia, on mietittävä mikä asia meni vikaan ja miten suunnitelmaa voitaisiin parantaa. Sekä onnistumisesta, että epäonnistumisesta opittua tietoa kannattaa hyödyntää prosessin jatkokehityksessä. Näitä neljä vaihetta voidaan toistaa yhä uudelleen osana jatkuvan oppimisen ja parantamisen päätymätöntä sykliä. On tärkeitä, että prosessia seurataan, mitataan ja kehitetään jatkuvasti, jolloin näiden vaiheiden toteutumisessa parannetaan sekä datan laatua, että datan laadun parantamisen prosesseja. (Deming Institute 2023; Väre 2019, 224–225.)

Datan laadun hallinta on moniulotteinen prosessi. Datan laadun parantamiseen on olemassa erilaisia työkaluja ja menetelmiä. Kuitenkin tärkeintä on ymmärtää, mikä on hyvä datan laatu juuri kyseiselle yritykselle, mitkä prosessit ja tekijät vaikuttavat datan laatuun. Väre (2019, 231) toteaa, että ei ole olemassa yleistä reseptiä datan laadun ongelman ratkaisemiseen, mutta yritysten kannattaa paneutua ainakin datan hallintamalliin, prosesseihin ja elinkaareen sekä kehittää omat datastandardit. Hyvää datan laatua voi ylläpitää vain integroimalla jatkuvaa master datan hallintaa organisaation päivittäistoimintoihin.

2.3.5 Nimikehallinnan keskeiset haasteet master datan hallinnan kannalta

Nimikedata on master dataa ja kaikki yllä mainitut laatuongelmat ja datan laadun parantamisen työkalut kuuluvat nimikehallintaan. Nimikehallinnan keskeiset haasteet ja ongelmat johtuvat pääosin datan laatuvirheistä. Ne voisi käsitellä datalaadun ulottuvuuksien kannalta.

Nimikedata ei ole kattava (puutteellinen), kun järjestelmien attribuuttikentistä puuttuu tarvittavat tiedot. Datan puutteellisuus voi johtua muun muassa käsittelijän tietymättömyydestä tai huolimattomuudesta, eri järjestelmien erilaisista vaatimuksista (esimerkiksi PDM:ssa joku tieto ei ole pakollinen, mutta ERP-järjestelmässä on). Nimikedatan puutteellisuus vaikeuttaa tai tekee mahdottomaksi hakutoimintoja ja nimikkeiden siirtoja järjestelmästä toiseen.

Nimikedatan oikeellisuus tarkoittaa, että kaikki järjestelmiin tallennettu nimiketieto pitää paikkansa. Virheet nimikedatassa voivat syntyä huolimattomuudesta datan syöttämisessä tai silloin, kun esimerkiksi samankaltaisen nimikkeen luomisvaiheessa data kopioidaan mallinimikkeestä. Myös data muuttuu virheelliseksi, kun se ei ole ajanmukainen, eli tuotteissa tapahtuvia muutoksia ei päivitetä järjestelmään. Virheellinen data voi aiheuttaa monia ongelmia kuten väärien tuotteiden tilaaminen, varastotietojen tai raportoinnin vääristyminen.

Datan ulottuvuudet ovat riippuvaisia toisistaan. Näin puuttuva tai virheelleen nimikedata vaikuttaa negatiivisesti nimikkeiden ainutlaatuisuuteen. Jos tarvittavaa nimikettä ei löydy järjestelmästä, luodaan uusi nimike, vaikka kyseistä tuotetta vastaava nimike olisi jo olemassa, mutta tietojen puutteellisuuden vuoksi sitä ei vain osattu hakea. Näin syntyvät nimikkeiden kaksoiskappaleet eli duplikaatit, ja tämä on nimikehallinnan iso haaste. Silloin kun duplikaattinimikkeet havaitaan, kummallakin nimikkeellä voi olla tapahtumahistoria ERP-järjestelmässä, omat varastopaikat ja saldot, ne voivat olla käytössä eri tuoterakenteissa (BOM:eissa). Tämän virheen voi korjata sulkemalla toisen nimikkeen, jolloin vain toinen jäisi käyttöön. Mutta asia ei aina ole niin yksinkertainen. Nimikkeiden varastosaldot pitää korjata sekä järjestelmässä, että fyysiseesti siirtämällä tuotteet samalle paikalle. BOM:ien korjaaminen voi olla vielä haasteellisempaa, jos kumpaakin nimikettä on käytetty monissa rakenteissa. Korjaustyöt vaatisivat useita tunteja suunnittelutyötä, mikä aiheuttaisi lisäkustannuksia, eikä siihen aina edes löydy riittävästi aikaa tai resursseja. Joskus nimikkeellä voi olla enemmän kuin yksi duplikaatti, silloin virheen korjaamisesta tulee vielä suurempi haaste.

Nimikedatan yhteneväisyysongelmat voivat johtua siitä, että järjestelmien välinen integraatio ei toimi hyvin ja tiedot eivät päivity kaikkiin järjestelmiin tai pahimmassa tapauksessa järjestelmien välissä ei ole tehty integraatiota ja nimiketiedot täytetään järjestelmiin käsin. Erässä työelämän tilanteessa yrityksen nimikkeet luotiin PDM:ään sekä ERP:iin käsin. Vuosien jälkeen yrityksessä otettiin uuden ERP-järjestelmän käyttöön. Uusi järjestelmä integroitiin PDM-järjestelmään, josta oli tullut nimiketiedon master, eli kaikki nimikkeet luotiin PDM:ään ja siirrettiin uuteen ERP:iin integraation kautta. Tämä käynnisti nimikedatan suursiivousprojektin. Ensinnäkin iso osa vanhoista nimikkeistä ei ollut valmis siirtoon uuteen järjestelmään vaatimustenmukaisuuden kannalta. Osalta nimikkeistä puuttui ERP:ssä pakolliset attribuutit tai nimikkeiden tila ei ollut kelvollinen siirtoon. Näiden nimikkeiden siirto ei olisi ollut mahdollinen ilman korjaustoimenpiteitä. Toiset nimikkeet olisivat olleet muodollisesti kunnossa, mutta sisällöllisesti olivat käyttökelvottomat uudessa ERP:ssä. Esimerkiksi tuoterakenteet, jotka oli luotu manuaalisesti tai tuotu CAD-ohjelmasta vanhaan ERP:iin, puuttuivat täysin PDM-järjestelmästä. Toisin sanoen, vaikka nimike oli teknisesti kunnossa ja se olisi ollut mahdollista siirtää ERP:iin, siitä ei olisi mitään hyötyä ilman BOM-rakennetta. Tai toisessa tapauksessa nimikedata oli virheellinen, koska nimikkeiden attribuutteja ei ole päivitetty PDM:aan. Luonnollisesti jokaista nimikettä ei ollut mahdollista käydä läpi ja iso määrä muodollisesti kelvollisia nimikkeitä ajettiin massana uuteen järjestelmään. Nimikkeiden mukaan siinne siirtyivät datan yhteneväisyyden ongelmat. Kahteen järjestelmään käsin täytetty data ei millään voinut olla täysin identtinen kummassakin järjestelmässä. Siihen vaikuttivat mahdolliset näppäilyvirheet, järjestelmien eri muodollisuusvaatimukset (eri attribuuttikentät, kenttien pituus oli erilainen) sekä tietojen epätasaiset päivitykset. Joten

PDM:sta siirretty nimiketiedot välillä erottuivat paljonkin vanhan ERP:n tiedoista. Tämä tuotti ongelmia ostajille komponenttihankinnan yhteydessä sekä kaikille muille järjestelmäkäyttäjille, jotka käyttivät ensisijaisesti ERP-järjestelmän nimiketietoja. Näihin ongelmiin ei ollut muuta ratkaisua, kuin tehdä nimikedatasta oikea ja yhteneväinen. Mutta virheellisten nimikkeiden löytäminen ja tietojen korjaus veivät todella paljon työaikaa, joka nosti yrityksen kustannuksia.

Nimikedata ei ole ajantasainen, jos sitä ei päivitetä ajan tasalle. Tämä aiheuttaa muun muassa elinkaareen liittyviä ongelmia. Nimikkeet poistuvat käytöstä eri syistä: tiettyjen komponenttien valmistus ja myynti loppuvat tai tuotemuutoksen vuoksi komponenttia ei enää tarvita yrityksen tuotannossa ja varaosamyynnissä. Jos käytöstä poistunut nimike on edelleen aktiivinen yrityksen tietojärjestelmissä, nimiketiedon käsittelijä voi käyttää sitä esimerkiksi uuden tuotteen suunnittelussa tai lisäämällä kyseisen osan osto- tai myyntitilauksen. Ennen kuin virhe havaitaan, siitä usein ehtii muodostua negatiivisia vaikutuksia ja lisäkustannuksia, kuten ylimääräinen selvittely komponenttitoimittajan kanssa tai tuoterakenteiden korjaaminen. Siksi on tärkeitä huolehtia nimikkeiden elinkaaren hallitusta päättymisestä. Tiedot käytöstä poistuvista nimikkeistä voi saada komponenttitoimittajilta, yrityksen eri sidosryhmiltä tai seuraamalla nimikkeen tapahtumahistoriaa ERP-järjestelmässä.

Kansainvälisesti toimivissa yrityksissä nimikedataa usein ylläpidetään usealla eri kiellellä, joten kaikki yllä mainitut nimikedatan sisältöä koskevat haasteet moninkertaistuvat, koska nimiketiedon on oltava kattava, oikea, ainutlaatuinen, vaatimustenmukainen, yhteneväinen sekä ajantasainen kaikilla kieliversioilla.

Nykydigimaailmassa datan hallinnan suurin haaste on datan määrä ja sen määrän jatkuva kasvu. Hyvän datan laadun ylläpitämiseen datan (nimikedatan mukaan lukien) kasvun on oltava hallittua. Viime aikoihin asti monet yritykset keskittyivät vain omaan ydintoimintaan, eikä datahallintaan ole panostettu tarpeeksi. Vasta silloin, kun huononlaatuisesta datasta muodostuivat suuret haasteet yrityksen liiketoiminnalle, alkoi tulla ymmärrys siitä, miten tärkeästä aiheesta on kyse. Vuosien saatossa yrityksille on kertynyt valtavasti kaikenlaista tietoa, joka on peräisin eri lähteistä, eri henkilöiden syöttämänä, jota ei päivitetty riittävän usein tai ei koskaan. Jotta yritys voisi jatkossa toimia joustavasti ja tehdä luotettavat päätökset laadukkaan datan perusteella, data on saatava kuntoon. Ei ole yhtenäistä sääntöä, miten usein suursiivousoperaatiot kannattaa tehdä; tämä riippuu muun muassa yritysten koosta, toimialalta, tietojärjestelmien ominaisuuksista. Kuitenkin nimikedata on yleensä pakko käydä läpi ainakin silloin, kun uusi tietojärjestelmä otetaan käyttöön tai siirretään nimikkeet vanhasta järjestelmästä uuteen. Työelämässä on tullut vastaan eri tapoja toteuttaa nimikemigraatiota. Toisessa yrityksessä ajan ja resurssien puutteen vuoksi oli päätetty

siirtää kaikki nimikkeet vanhasta ERP:sta uuteen ja siivota data vasta uudessa järjestelmässä. Silloin saatiin uusi järjestelmä käyttöön suhteellisen nopeasti, mutta nimikedatan siivousprojekti on kestänyt useita vuosia käyttöönoton jälkeen. Toisessa yrityksessä tarkoituksena oli siivota mahdollisimman paljon nimikkeitä ennen siirtoa uuteen ERP:iin ja vain osa nimikkeistä siirrettiin massana. ERP:n käyttöönoton jälkeen kaikki järjestelmästä puuttuvat nimikkeet siirrettiin PDM:sta ERP:iin yksitellen manuaalisesti nimiketietojen tarkistuksen jälkeen nimikehallinnan toimesta. Molemmissa tapauksissa on omat plussat ja miinukset, mutta kumpikin esimerkki osoittaa, että ison hallitsemattomasti muodostuneen datamassan jälkikäsittely vaatii paljon työaikaa ja osaamista saadakseen tiedot oikeaksi ja tarkoituksenmukaiseksi. Datasiivouksen positiiviset vaikutukset eivät kuitenkaan kestä pitkään, jos keskitetystä nimikehallinnasta ei tehdä osaa yrityksen toimintaa. Ainakin vastuu nimikkeiden avaamisesta ja sulkemisesta pitäisi olla keskitetty tietylle osastolle tai henkilöille. Yrityksellä on oltava selkeä säännöstö, jonka mukaan nimikkeet luodaan, nimetään ja täytetään attribuuttikentät. Ennen uuden nimikkeen avaamista tulee aina tarkistaa, onko kyseinen tuote jo olemassa järjestelmässä. On huolehdittava, että nimiketieto päivittyy kaikkiin järjestelmiin samanaikaisesti ja käytöstä poistuvien nimikkeiden status on näkyvissä kaikkialla. Jos nimiketiedot ylläpidetään eri kielillä, on hyvä luoda master järjestelmään sanasto ainakin usein käytetyistä termeistä ja käsitteistä. Tämä auttaa vähentämään mahdollisten nimikekuvausten käännösvaihtoehtojen määrän ja tekee nimikekuvaukset yhteneväisemmiksi. Keskitetyn nimikehallinnan avulla ratkaistaan monet nimikehallinnan ongelmat, mutta järjestelmätyöntekijöiden lisäksi yrityksen nimikkeitä tyypillisesti käsittelevät eri henkilöt ja sidosryhmät, kuten suunnittelu, tuotanto, osto, myynti, varasto tai asiakaspalvelu. Jokainen ryhmä käsittelee nimikkeitä omasta näkökulmastaan, käyttää ja päivittää niihin liittyvää dataa eri tarkoituksiin. Jos järjestelmien ja datan käyttöön liittyvät pelisäännöt eivät ole tarkkaan määritelty ja informoittu läpi organisaation tai puuttuvat kokonaan, toisen henkilön tai ryhmän toimintatavat (huolimattomuudesta, tietämättömyydestä tai motivaation puutteesta johtuen) voivat vaikeuttaa muiden käyttäjien työtä ja vaikuttaa negatiivisesti dataan laatuun. Toisessa työelämän tilanteessa myyntiosasto on myynyt jatkuvasti samaa tuotetta samalle asiakkaalle, joten uuden tilauksen luomisen sijaan oli helpompaa kopioida vanhan myyntitilauksen ja päivittää sinne vain tuotteiden määrää. Tilausten välisenä aikana nimikkeen varastopaikka on saattanut muuttua. Koska vanhat varastotiedot olivat kopioitu uudelle tilaukselle, järjestelmä yritti hakea tuotetta vanhasta varastopaikasta, jossa sitä ei ollut. Jos varastossa olisi huomattu ja korjattu virhe, korjaus olisi vaatinut vain varaston työntekijöiden työaikaa, mutta jos virhe olisi jäänyt huomaamatta, ”puuttuvasta” tuotteesta voitaisiin tehdä osto- tai tuotantotilaus, riippuen siitä oliko kyseessä ostettava vai valmistettava tuote. Silloin virheen negatiiviset vaikutukset olisivat olleet paljon suurempia. Joten pelkkä säännöstöjen ja standardien luominen ei riitä, on yhtä tärkeää kommunikoida ja

informoida uusista käytännöistä sekä antaa tarvittu tuki ja koulutus kaikille tietojen käsittelijöille. Tietoisuus ja ymmärrys motivoi ihmisiä toimimaan oikein. Järjestelmät ovat vain työkaluja ihmisten käsissä, jotka oikein käytettyinä helpottavat ihmisten toimintaa paljon. Yrityksen luomat datahallintatavat, prosessit ja standardit sekä työntekijöiden tietoisuus ja motivaatio ovat huomattavasti isommassa roolissa onnistuneen datahallinnan toteuttamisessa.

3 Nimikedatan laadunvarmistus

3.1 Tutkimusdata ja tutkittavat ongelmatapaukset

Opinnäytetyön tutkimusosuudessa käytetään asiakasyrityksien PDM- sekä ERP järjestelmistä saatua nimikedataa Excel-tiedoston muodossa. Tutkimustapauksesta riippuen nimikekoodin ja kuvauksen lisäksi Exceliin saadaan erilaisia nimikeattribuutteja.

Kuten johdannossa mainittiin, tämän työn tavoitteena on löytää automatisoidut ratkaisut tiettyjen ongelmatapausten havaitsemiseen nimikemassasta. Tutkittavat ongelmatapaukset ovat nimikkeiden kaksoiskappaleet sekä nimikkeet, joiden attribuuteissa on puutteita.

3.2 Nimikkeiden kaksoiskappaleet

Tämän käyttötapauksen nimikedatasetit ajettiin asiakkaan PDM-järjestelmästä tuoteryhmän perusteella. Tutkittavan nimikemassan läpikäynnin jälkeen oli mahdollista havaita erityyppisiä duplikaattinimikkeitä:

- Täydelliset duplikaatit: Kokonaan tai lähes täysin samanlaiset nimikkeet, näitä tapauksia on kohtuullisen helppoa löytää.
- Piiloduplikaatit: Nimikkeet, joilla on osittain samat, mutta osittain eri tai eri lailla täytetyt attribuutit. Niiden löytäminen vaatii datan perusteellista tutkintaa tarkistettavien attribuuttien sekä käytettyjen ratkaisujen valitsemiseksi.
- Nimikkeiden kaksoiskappaleet, jotka eivät ole duplikaatteja järjestelmämielessä, esimerkiksi samalle tuotteelle tehdyt nimikkeet, joilla voi olla eri valmistaja; myös nimi tai tuotenumero voivat erota. Näiden tapausten havaitsemiseen on mahdotonta kehittää automatisoitua ratkaisua. Tähän tehtävään tarvitaan kyseisen tuoteryhmän asiantuntijan osaamista, siksi tämänkaltaisten tapausten tutkiminen on jätetty opinnäytetyön ulkopuolelle.

Duplikaattinimikkeiden tutkimukseen valittiin 2 datasettiä: sähkökomponenttinimikkeet sekä ruuvinimikkeet. Kumpikin tuoteryhmä sisältää monissa kokonaisuuksissa käytettyjä standardituotteita, joilla voi olla useita valmistajia ja toimittajia, siksi kaksoisnimikkeet usein esiintyvät näissä tuoteryhmissä. Kummatkin Excel-tiedostot sisälsivät yli 2000 nimikeriviä. Tutkimuksen tavoitteena oli löytää ratkaisuja sekä täydellisten että mahdollisesti niin sanottujen piiloduplikaattien löytämiseen.

3.2.1 Täydelliset nimikeduplikaatit

Täydellisten duplikaattien löytämistä kokeiltiin kahdella eri työkalulla: sekä Excel-kaavojen avulla että koneoppimismallia ja Python-ohjelmointikieltä käyttäen. Excel-ratkaisun hyvät puolet ovat helppokäyttöisyys ja riippumattomuus ohjelmointialustasta. Ratkaisu ei vaadi koodausta ja sitä pystyy käyttämään Excelin työpöytäsovelluksella ja pilviversiolla. Tämä ratkaisu voi hyvin toimia pienimmille dataseteille tai silloin, kun yrityksellä ei ole resursseja ohjelmointiratkaisujen kehittämiseen. Isommille dataseteille olisi kuitenkin kätevämpää käyttää koneoppimisratkaisua. Sen kehitys ja testaaminen vievät huomattavasti enemmän aikaa, mutta kun oikeat parametrit on löydetty, ratkaisun käyttö on helppoa ja joustavaa, ja koodia voi muokata lähtödatan mukaan. Ratkaisu mahdollistaa myös duplikaattinimikkeiden ja niille valittujen attribuuttien ajamisen erillisen tiedostoon, mikä helpottaa tulosten tutkimista ja esittämistä.

Nimikeduplikaattien löytäminen Excelin avulla

Jos nimikkeiden attribuuttikentät olisi täytetty yhteisen standardin tai säännöstön mukaan, selkeiden duplikaattien löytämiseen ei olisi tarvittu muita toimintoja kuin nimikekuvaus-sarakkeen läpikäynti Conditional formating →Duplcate values työkalun avulla. Kuitenkin tutkimusdata sisälsi paljon vanhoja nimikkeitä, joiden attribuuttikenttiä olivat täyttäneet eri ihmiset eri aikoihin, usein ilman yhtenäistä logiikkaa. Joten paremman tuloksen saamiseksi tarkistukseen olisi järkevä lisätä ainakin toinen attribuutti. Tässä tapauksessa duplikaatit etsittiin spesifikaation (Specification, esimerkin E-sarake) ja nimikekuvauksen (Name, esimerkin B-sarake) sarakkeista nimenomaan tässä järjestyksessä. Spesifikaatio-kenttä yleensä antaa enemmän yksilöivää tietoa nimikkeestä, joten ensin etsittiin duplikaatit aakkosjärjestykseen mukaan lajitellusta Specification-sarakkeesta, jonka jälkeen tarkistettiin, onko kyseisellä nimikeparilla samat kuvaukset, ja jos oli, merkattiin nimikkeet duplikaateiksi (taulukko 4).

Number	Name	Description (English)	Description (Finnish)	Specification	Duplicate check	Excel Formula for duplicate checking
0123456	HEX. SCREW M10 x 50 m 8.8	Hexagon screw	Kuusioruuvi	M10x50 - 8.8		
1000123	HEX. SCREW M10 x 50 m 8.8	Hexagon screw	Kuusioruuvi	M10x50 - 8.8	duplicate	=IF(AND(E2=E3;B2=B3);"duplicate";"")

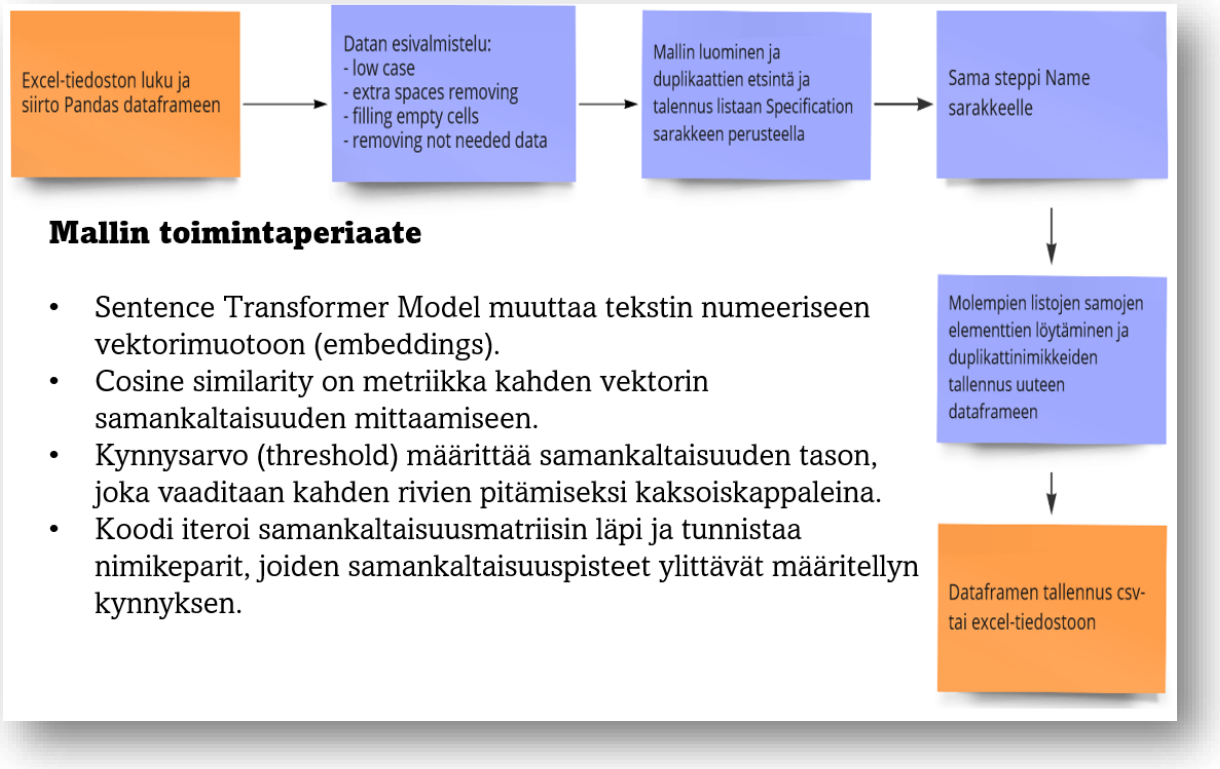
Taulukko 4. Duplikaattinimikkeiden havaitseminen Excel-kaavion avulla

Excelin avulla löydettiin useita kymmeniä duplikaattipareja nimiketiedostoista.

Nimikeduplikaattien löytäminen Sentence Transformers mallia käyttäen

Koneoppimisratkaisussa nimikeduplikaatit etsittiin Pythonin SentenceTransformer kirjaston esikoulutetun mallin avulla. Ensimmäinen Sentence Transformer malli oli julkaistu vuonna 2017 ja siitä lähtien NLP (Natural Language Processing) käsittely on muuttunut huomattavasti (Pinecone 2021). Transformerit ovat perinteisten RNN-mallien epäsuoria jatkajia. Kuten RNN-verkot, ne muuttavat tekstin numeeriseen vektorimuotoon (embeddings). Toisin kuin perinteiset mallit, jotka koodaavat jokaisen sanan erikseen, Sentence Transformers ottavat huomioon lauseen kokonaisuudessaan. Tämä auttaa niitä ”ymmärtämään” sanojen väliset suhteet ja merkitykset koko lauseen yhteydessä. Trasformerit käyttävät itsetarkkailumekanismeja, joiden avulla malli voi punnita syötetyn tekstin osia vektorien luomisvaiheessa. Monet Sentence Transformers mallit ovat esikoulutettuja suurilla tekstikokonaisuuksilla käyttäen ohjaamatonta tai ohjattua koneoppimista. Transformereiden merkittävä sovellus on lauseiden tai tekstien semanttisen samankaltaisuuden mittaaminen. Kontekstivektoreiden avulla malli pystyy analysoimaan tekstin semanttisen sisällön, mikä mahdollistaa tarkan vertailun ja samankaltaisuuden pisteytyksen. Tämän lisäksi Sentence Transformers ovat resurssitehokkaita. Ne tuottavat pienempiulotteisia vektoreita verrattuna monimutkaisempiin malleihin. Tämä vähentää muisti- ja laskentakapasiteettivaatimuksia ja tekee malleista sopivia resurssirajoitteisiin ympäristöihin. Transformerit voivat saavuttaa hyvän suorituskyvyn pienemmällä määrällä näytteitä, mikä tekee niistä tehokkaita, kun dataa on rajoitetusti. Esikoulutetuilla malleilla on yleensä nopeammat vasteajat, joten ne sopivat reaaliaikaisiin sovelluksiin tai skenaarioihin, joissa tarvitaan nopeita vastauksia. Pienempien resurssitarpeidensa vuoksi malleja voidaan käyttää useammille laitteille, joiden laskentateho on rajoitettu. Monet transformerit on suunniteltu käsittelemään useita kieliä. Tämä tekee niistä monipuolisia sovelluksiin, jotka vaativat tekstin käsittelyä eri kielillä ja lisää niiden käyttökelpoisuutta erilaisissa kieliympäristöissä. (Mudadla 2023).

Tässä työssä käytetiin SentenceTransformer kirjaston mallia *paraphrase-MiniLM-L6-v2*, jota on kehitetty englanninkielisen tekstin käsittelyyn. Kuvio 8 esittää duplikaattihakuprosessia ja mallin toimintaperiaatteet. Duplikaattien hakulogiikka on suunnilleen samanlainen kuin Excel-haussa: ensin etsitään duplikaatit Specification- sarakkeesta, sitten Name-sarakkeesta ja lopputuloksena on setti riveistä, joita lasketaan duplikaateiksi kummassakin tarkistuksessa.



Kuvio 8. Duplikaattinimikkeiden havaitseminen SentenceTransformer mallin avulla

Ensimmäinen vaihe on Excel-tiedoston muodossa olevan datan luku (kuva 1). Datan muotoa muutetaan csv-tiedostoksi ja sitten tallennetaan pandas-dataframeen.

```
import pandas as pd
# reading data
# Read and store content of an excel file
read_file = pd.read_excel('electric_components.xlsx')
# Write the dataframe object into csv file
read_file.to_csv ("electric_components.csv", index = None, header=True)
# read csv file and convert into a dataframe object
df = pd.DataFrame(pd.read_csv("electric_components.csv"))

df
```

Kuva 1. Datan luku Excel-tiedostosta

Seuraavaksi data tulee esivalmistella jatkokäsittelyn helpottamiseksi ja tulosten tarkkuuden parantamiseksi. Tämä vaihe on esitetty kuvassa 2. Tyhjät solut täytetään none-arvolla, tekstin kirjaimet muutetaan pieniksi (low case), poistetaan tekstistä ylimääräiset välilyönnit sekä poistetaan dataframesta tarpeettomat sarakkeet ja obsolete-tilassa olevat nimikerivit.

```
# preprocessing data
lowerify_cols = [col for col in df if col not in ['Number']]
# low case to text columns except Item number
df[lowerify_cols]= df[lowerify_cols].apply(lambda x: x.astype(str).str.lower(),axis=1)
df.columns = df.columns.str.strip() # removing heading and trailing spaces
df.columns  = df.columns.str.replace(' ','')# removing extra cpaces in text
df = df.fillna('none')# filling empty cells
# dropping lines with obsolete items
df.drop(df[df['State'] =='obsolete'].index, inplace=True)
#dropping columns that are not needed
df.drop(['Description(Finnish)', 'State', 'MaterialGroup', 'ItemTechnicalNote(FI)'],
        axis=1, inplace=True)
df.reset_index(drop=True, inplace=True) # reseting index
```

Kuva 2. Datan esikäsittely

Esikäsittelyn jälkeen ladataan tarvittavat kirjastot ja rakennetaan malli (kuva 3). paraphrase-MiniLM-L6-v2 malli muuttaa Specification sarakkeen tekstin numeeriseen vektorimuotoon (embeddings). Vektorien samankaltaisuuden mittaamiseen käytetään Cosine similarity metriikka. Sen avulla rakennetaan samankaltaisuusmatriisi (similarity matrix), joka sisältää mallin tuottamat vektorit. Kynnysarvo (threshold) määrittää samankaltaisuuden tason, joka vaaditaan kahden rivien pitämiseksi kaksoiskappaleina. Tässä tapauksessa käytettiin kynnysarvoa 0.999, koska tarkoituksena oli hakea täydelliset duplikaatit. Koodi iteroi samankaltaisuusmatriisin läpi ja tunnistaa kuvausparit, joiden samankaltaisuuspisteet ylittävät määritellyn kynnyksen. Duplikaattirivien tiedot tallennetaan listaan.

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
# building model
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')
# searching for duplicate pairs in Specification column
# generating embeddings (numerical representations of text)
# using model for Specification column contents
embeddings = model.encode(df['Specification'].tolist())
df['embeddings'] = embeddings.tolist()

# Cosine similarity measures the similarity between two vectors (embeddings)
# Similarity matrix provides a comprehensive overview of how similar each sentence
#is to every other sentence in the dataset
similarity_matrix = cosine_similarity(embeddings, embeddings)
duplicates = []
# iterating through similarity matrix finding duplicate pairs storing them in a list of tuples
for i in range(len(similarity_matrix)):
    for j in range(i + 1, len(similarity_matrix)):
        # threshold value determines the level of similarity considering two sentences as duplicates
        if similarity_matrix[i][j] > 0.999:

            # storing duplcate items' parameters ito a list
            duplicates.append((df['Number'][i], df['Name'][i], df['Specification'][i],
                               df['Number'][j], df['Name'][j], df['Specification'][j]))

print(len(duplicates))
```

Kuva 3. Datan käsittely koneoppimismallin avulla

Tämä vaihe toistetaan Name-sarakkeelle, minkä jälkeen yhdistetään molempien listojen samat elementit uuteen dataframeen. Lopuksi tallennetaan tulokset csv- tai Excel-tiedostoon (kuva 4).

```
# finding identical duplicate pairs in two lists (Specification and Name column duplicates)
all_duplicates = set(duplicates) & set(duplicates1)
print(len(all_duplicates))

# Create DataFrame for Duplicate Pairs
duplicate_df = pd.DataFrame(all_duplicates, columns=
                            ['Number', 'Item name', 'Specification','Duplicate number',
                             'Duplicate name', 'Duplicate specification'])

# Print the DataFrame
duplicate_df

# saving result as csv (or excel) file
#duplicate_df.to_excel('duplicate_electric.xlsx', index=False)
duplicate_df.to_csv('duplicate_electric.csv', index=False)
```

Kuva 4. Duplikaattihakutulosten tallennus csv- tai Excel-tiedostoon

Tämän menetelmän avulla löydettiin kymmeniä pareja duplikaattinimikkeitä. Reilun kahden tuhannen rivin käsittely kesti muutamia minuutteja. Koodi on toiminut yhtä hyvin sekä ruuvinimikkeille että sähkökomponenteille. Ainoastaan ruuvinimikeduplikaattien tarkistuksen yhteydessä selvisi, että samankokoiset osa- ja täysikierteiset ruuvit, joiden standarditieto puuttui Specification tai Name kentistä oli laskettu duplikaateiksi. Eli tässä tapauksessa olisi pitänyt ottaa Standard kenttää mukaan duplikaattitarkistukseen. Tämä on osoitus siitä, miten tärkeä on tutustua tutkittavaan dataan huolellisesti ja miten eri attribuutit voivat olla eriarvoisia tuoteryhmästä riippuen.

3.2.2 Piiloduplikaatit

Niin sanottujen piiloduplikaattien löytämisessä käytettiin Regular Expressions metodia ja Python koodausta. Regular Expressions (regex) eli säännölliset lausekkeet on erittäin tehokas työkalu tekstimerkkijonojen etsimiseen ja manipulointiin, erityisesti tekstitiedostojen käsittelyssä. Yksi regex-rivi voi helposti korvata useita kymmeniä rivejä ohjelmointikoodia. Regexiä tukevat kaikki skriptauskielet esimerkiksi Perl, Python, PHP ja JavaScript, sekä monet yleisohjelmointikielet ja jopa tekstinkäsittelyohjelmat, kuten Word. (Nanyang Technological University 2018).

Pythonissa regexiä tukee *re* -moduuli ja regex haku yleensä näyttää tältä:

match = re.search(pat, str)

re.search()-funktio ottaa sisään regex mallin (pattern) sekä merkkijonon (string) ja etsii mallia vastaavaa tekstiä merkkijonosta. Jos haku onnistuu, *search()* palauttaa match-objektin, jos ei, ei palauteta mitään. Siksi hakua seuraa yleensä jos-lauseke, jolla testataan, onnistuiko haku. (Google for Education 2023.)

```
import re

str = 'an example word:cat!!'
match = re.search(r'word:\w\w\w', str)
# If-statement after search() tests if it succeeded
if match:
  print('found', match.group()) ## 'found word:cat'
else:
  print('did not find')
```

Kuva 5. Regex-tekniikan käyttöesimerkki (Google for Education 2023)

Kuvan 5 esimerkissä etsitään merkkijonoa ”world:”, jotta seuraa kolmen kirjaimen sana. Koodi tallentaa hakutuloksen muuttujaan ”match”. Sitten if-lausekkeella testataan, joko hakutulos on tosi, silloin haku onnistui ja *match.group()* on vastaava teksti (tässä tapauksessa sana ”cat”). Muuten hakutulos on epätosi, silloin haku ei onnistunut ja mallia vastaavaa tekstiä ei löytynyt. "r" mallimerkkijonon alussa tarkoittaa Pythonin "raakaa" merkkijonoa ilman muutoksia. Esimerkissä käytetty *re.search()*-funktio löytää vain ensimmäisen vastaavuuden merkkijonosta. Kuitenkin isojen datakokonaisuuksien ja tiedostojen läpikäyntiin käytetään ehkä re-moduulin tehokkain funktio *findall()*, joka etsii kaikki mallia vastaavat osumat ja palauttaa ne merkkijonolistana, jossa jokainen merkkijono edustaa yhtä osumaa (kuva 6). (Google for Education 2023.).

```
## Suppose we have a text with many email addresses
str = 'purple alice@google.com, blah monkey bob@abc.com blah dishwasher'

## Here re.findall() returns a list of all the found email strings
emails = re.findall(r'[\w\.-]+@[\w\.-]+', str) ## ['alice@google.com', 'bob@abc.com']
for email in emails:
  # do something with each found email string
  print(email)
```

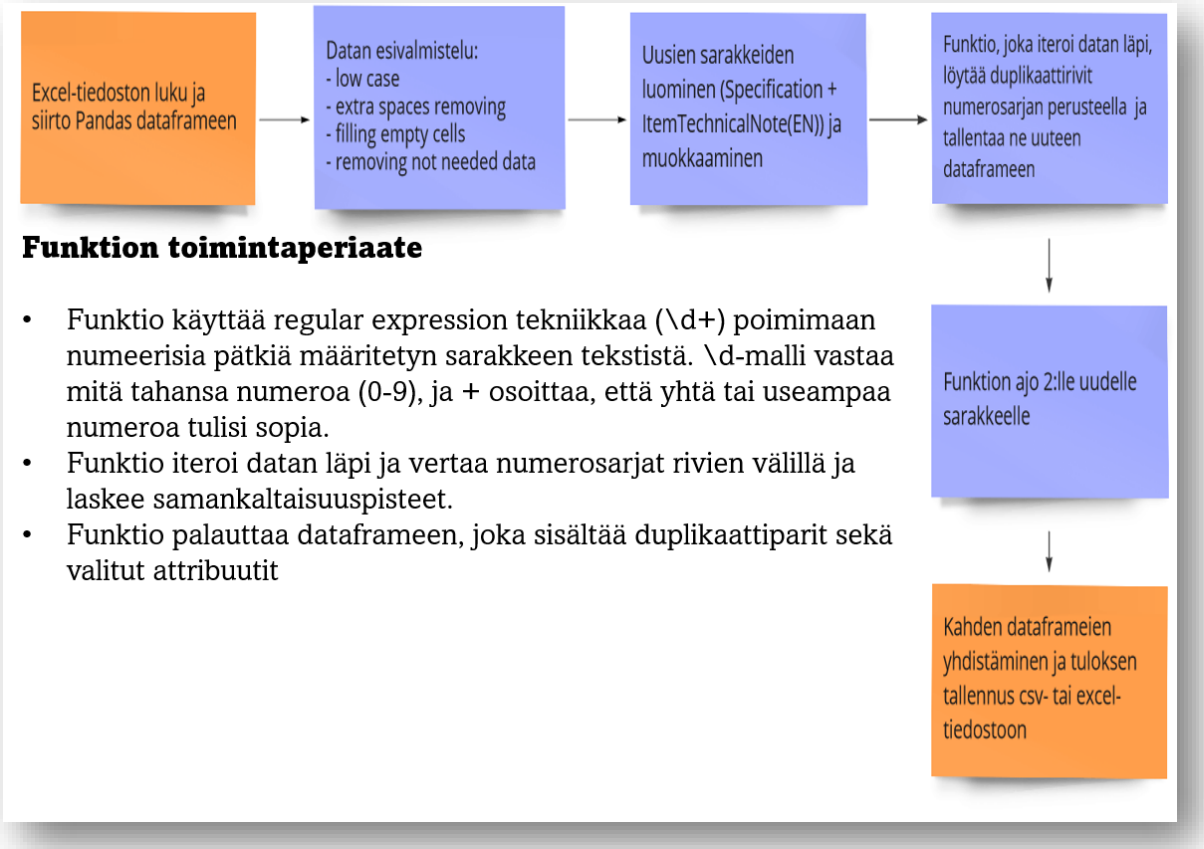
Kuva 6. *findall()* funktion käyttö sähköpostiosoitteiden etsimisessä (Google for Education 2023)

Säännölliset lausekkeet ovat erittäin tehokkaita myös siksi, että niiden avulla voidaan hakea tiettyä mallia vastaavaa tekstiä, eikä pelkästään kiinteitä merkkejä. Alla muutama esimerkki hakumalleista:

- \w (pienellä) - vastaa kirjainta, numeroa tai alaviivaa [a-zA-Z0-9_]

- \W (isolla) - vastaa mitä tahansa muuta kuin sanamerkkiä
- \d- desimaaliluku [0–9]
- \s (pienellä) - vastaa yhtä tyhjää merkkiä – esim. välilyöntiä, rivinvaihtoa, sarkainta
- \S (isolla) - vastaa mitä tahansa muuta kuin välilyöntinä olevaa merkkiä
- ^ = alku, $ = loppu - vastaa merkkijonon alkua tai loppua. (Google for Education 2023).

Piiloduplikaattihakua varten oli löydettävä yhdistävä tekijä, jonka perusteella nimikkeet voisi laskea duplikaateiksi. Sähkökomponenteilla se oli tuotteiden sähkönumero. Regular Expressions metodin avulla nimikekuvaustekstistä etsittiin vähintään 7-merkkiset numerosarjat ja vertailtiin nimiketiedoston rivit, joilta ne löydettiin. Kuten aiemmin mainittiin, tutkittava nimikedata ei ollut kovin laadukasta, vanhojen nimikkeiden attribuutit oli täytetty ilman tarkkaa standardia, joten nimikkeiden sähkönumeroiden löytämiseksi piti luoda uusi sarake, yhdistämällä kaksi attribuuttisaraketta, joissa tämä tieto esiintyi useammin. Duplikaattien löytämiseksi tehtiin funktio, joka iteroi datan läpi, löytää samat numerosarjat tietystä sarakkeesta ja palauttaa duplikaattinimikkeet pandas dataframen muodossa. Datan käsittelyprosessi sekä funktion toimintaperiaatteet on esitetty kuviossa 9.



Kuvio 9. Piiloduplikaattinimikkeiden havaitseminen regex-tekniikan avulla

Datan luku ja esivalmisteluvaiheet tässä tapauksessa olivat lähes samanlaiset kuten täydellisten duplikaattien etsimisessä. Ainoa ero oli, että tarkistettavaksi otettiin sarakkeet Specification ja ItemTechnicalNote(EN), koska jompikumpi niistä yleensä sisälsi tuotteen sähkönumeron. Seuraavaksi luotiin uudet sarakkeet yhdistämällä ylämainitut attribuutit. Uudesta sarakkeesta Long spec joutui tekemään 2 versiota, toinen sisälsi vain 2 sarakkeiden yhdistetyn tekstin, toisesta yhdistämisen jälkeen piti ensin poistaa kaikki välilyönnit, sitten jaotella numeeriset ja ei-numeeriset merkkisarjat välilyöneillä. Tämän operaation avulla pystyttiin löytämään numerosarjat, jotka alun perin sisälsivät välilyöntejä, että voittaisi tunnistaa sarjat 10 20 30 ja 102030 samoiksi. Kuva 7 kuvaa tätä vaihetta.

```
# adding Long spec column joiniing Specification and ItemTechnicalNote(EN) columns
df['Long spec'] = df[['Specification', 'ItemTechnicalNote(EN)']].fillna('').agg(','.join, axis=1)
df["Long spec"] = df["Long spec"].str.replace("nan", "") #deleting nan-valuen from new column
# adding Long spec1 column, deleting all the spaces and then
# deviding strings with numbers and text in different strings
# for finding number sequences with spaces eg. 10 20 30 and 102030
df['Long spec1'] = df['Long spec'].str.replace(" ", "")
df['Long spec1'] = df['Long spec1'].str.replace(r'(\D+)(\d+)', r' \1 \2 ', regex=True)
```

Kuva 7. Attribuuttisarakkeiden yhdistäminen

Tämän jälkeen luotiin funktio, joka etsi samat numerosarjat datasarakkeesta ja palautti dataframe taulukon, joka sisälsi nimikeduplikaattiparit. Kokeilujen perusteella numerosarjan minimipituudeksi valittiin 7 ja samankaltaisuuden kynnysarvoksi 99. Funktio on esitetty kuvassa 8.

```
# function to find similar snippets with numbers in a column
# based on a threshold and minimum numeric length.
def find_similar_snippets_with_numbers_df(column, threshold=99, min_numeric_length=7):
    results = []
    # Regular expression pattern to extract numbers
    number_pattern = re.compile(r'\d+')

    # Iterate through each row in selected column
    for index, text in enumerate(column):
        # Extract numbers from the current text
        current_numbers = [int(match) for match in number_pattern.findall(text)
                           if len(match) >= min_numeric_length]

        # Compare the current numbers with the remaining numbers
        remaining_rows = df.iloc[index+1:]
        matches = []

        # Extract numeric snippets from the remaining text
        for _, remaining_text in remaining_rows.iterrows():
            remaining_numbers = [int(match) for match in number_pattern.findall(remaining_text[column.name])
                                 if len(match) >= min_numeric_length]
            intersection = set(current_numbers).intersection(remaining_numbers)

            if intersection:
                similarity_score = len(intersection) / max(len(current_numbers), len(remaining_numbers))
                if similarity_score >= threshold / 100:
                    match_index = remaining_text.name
                    results.append({
                        #'Row 1': index + 1,
                        'Item': df.at[index, 'Number'],
                        'Name': df.at[index, 'Name'],
                        'Specification': df.at[index, 'Specification'],
                        #'Row 2': match_index + 1,
                        'Similar item': df.at[match_index, 'Number'],
                        "Similar item's Name": df.at[match_index, 'Name'],
                        "Similar item's specification": df.at[match_index, 'Specification']
                        #'SimilarityScore': similarity_score
                    })

    # Create a new DataFrame from the results
    result_df = pd.DataFrame(results)

    return result_df
```

Kuva 8. Duplikaattihakufunktio

Funktio ajettiin kahdelle uudelle sarakkeelle ja saatiin kaksi duplikaattitaulukkoa, jotka yhdistettiin pudottamalla toistuvat rivit. Lopullinen dataframe tallennettiin csv- tai Excel-tiedostoon (kuva 9).

```
# Apply the function to the Long spec column
column_name1 = 'Long spec'
df1 = find_similar_snippets_with_numbers_df(df[column_name1])
df1
# Apply the function to the Long spec1 column
column_name2 = 'Long spec1'
df2 = find_similar_snippets_with_numbers_df(df[column_name2])
df2

# joining dataframes dropping similar entries
result_df = pd.concat([df1,df2]).drop_duplicates().reset_index(drop=True)
# Print the result DataFrame
#print(result_df)
result_df
# saving result as csv (or excel) file
# duplicate_df.to_excel('similar_electric.xlsx', index=False)
result_df.to_csv('similar_electric.csv', index=False)
```

Kuva 9. Funktion ajo, tulosten yhdistäminen ja tallennus

Tämän menetelmän avulla löydettiin 38 nimikeparia, joista 36 olivat selkeästi samalle tuotteelle tehdyt nimikkeet, vaikka niiden kuvaukset erosivat toisistaan. Kaksi nimikeparia oli laskettu virheellisesti duplikaateiksi, koska sisälsivät pitkiä nollista koostuvia numerosarjoja.

Tämä ratkaisu sopi hyvin sähkökomponenteille. Ruuvinimikkeille siitä ei kuitenkaan ollut suurta apua, koska tällä tuoteryhmällä ei ollut vastaavaa yhdistävää tekijää kuten tuotenumero, siksi koodin koeajo tälle datalle ei antanut käyttökelpoisia tuloksia. Tässäkin korostuu datan luonteen merkitys ja voidaan jälleen todistaa, että ei ole mahdollista löytää yhteistä ratkaisua kaikkiin ongelmiin. Tutkimalla dataa huolellisesti ja asettamalla oikeat kriteerit ja tavoitteet voidaan päättää mikä menetelmä toimisi parhaiten jokaisessa konkreettisessa tapauksessa.

3.3 Puutteellisen nimikedatan harmonisointi

Tämän työn kappaleessa 2.3.3 puhuttiin laadukkaan datan määritelmästä ja sen laadun mittaamisessa datan laadun ulottuvuuksien avulla. Yrityksen datastandardit ja säännöstöt määrittävät, mikä on hyvälaatuinen data kyseiselle yritykselle. Vertailemalla olemassa olevaa nimikedataa standardivaatimuksiin voidaan päättää, täytyykö tietyt datan laadun ulottuvuudet kuten esimerkiksi datan kattavuus, oikeellisuus, ainutlaatuisuus ja vaatimustenmukaisuus, toisin sanoen vastaako kyseinen data liiketoiminnan tarpeisiin. Jos yrityksen nimikedatahallinta on ollut hajanaista ilman tarkkoja standardeja ja säännöstöjä, datan laatu ei voi palvella yritysten toimintoja tarkoituksenmukaisesti. Siinä tapauksessa yrityksen on perustettava datastandardi, nimikedatan puutteet on selvittävä ja korjattava. Datahuoltoprojektit ovat yleensä laajoja ja vaativat paljon työtä (myös rutiinityötä) ja investointeja. Pitkässä juoksussa tämä työ kannattaa, koska kerran harmonisoitu ja jatkossa kunnossapidetty

nimikedata tukee yrityksen tuloksellista toimintaa ja kasvua sekä auttaa säästämään resursseja tulevaisuudessa.

3.3.1 Nimikedatan laadun mittausesimerkki

Datan laatua voi mitata monin eri tavoin. Alla on Python koodausta käyttäen toteutettu esimerkki datan laadun mittaamisesta edellisestä kappaleesta tutun sähkökomponenttinimikedatan perusteella. Nimikedatasäännöstön mukaan nimikkeen nimen (Name) on oltava kuvauksen (Description) ja spesifikaation (Specification) yhdistelmä. Vertailemalla Name-sarakkeen yhdistelmäsarakkeeseen Description + Specification, joka sisältää kahden edellä mainitun sarakkeen datan, voidaan päättää kyseisen nimikedatan laadusta. Ensin ladataan Excel-tiedoston data pandas dataframeen ja esivalmistellaan se (kuva 10).

```
# Data Quality check example
import pandas as pd
#reading and preprocessing data
# Read and store content of an excel file
df = pd.read_excel('fuzzy_check.xlsx')
lowerify_cols = [col for col in df if col not in ['Number']]
# low case to text columns except Item number
df[lowerify_cols]= df[lowerify_cols].apply(lambda x: x.astype(str).str.lower(),axis=1)
df.columns = df.columns.str.strip() # removing heading and trailing spaces
df.columns  = df.columns.str.replace(' ','') # removing extra cpaces in text

df.reset_index(drop=True, inplace=True)

print(df)
```

Kuva 10. Datan lataus ja esivalmistus ennen laatutarkistusta

Seuraavaksi vertaillaan kahden sarakkeen data ja lasketaan samankaltaisuusprosentti käyttäen Fuzzy String Matching metodia (kuva 11). Data myös jaetaan samankaltaisuusryhmiin (bin sarake), 20 % asteikolla niin, että ensimmäisessä ryhmässä ovat nimikkeet, joilla kahden vertailussa olevien sarakkeiden sisällöt ovat täysin identtisiä (100 %) toisessa ryhmässä datan samankaltaisuus on 80 %, kolmannessa 60 % ja niin edelleen.

```python
from fuzzywuzzy import fuzz
import numpy as np

# Comparing 2 columns data using Fuzzy String Matching
df['Similarity'] = df.apply(lambda x: fuzz.token_sort_ratio(x['Name'],
                     x['Description+Sepcification']), axis=1)

# Deviding data into bins based on similarity with a 20% step
df['bin'] = df['Similarity'].apply(lambda x: 20*np.floor(x/20))
df = df.sort_values(by=['Similarity'], ascending=False)
df.to_csv('similarity.csv')
df
```

	Number	Name	Description+Sepcification	Similarity	bin
1720	M00016585	protective hose fitting m16-b10-0 ana-quick profi	protective hose fitting m16-b10-0 ana-quick profi	100	100.0
1788	M00016802	terminal block pt10	terminal block pt10	100	100.0
1795	M00016809	plug-in bridge fbs4-6	plug-in bridge fbs4-6	100	100.0
1794	M00016808	plug-in bridge fbs3-6	plug-in bridge fbs3-6	100	100.0
840	3012081	cable carrier chain sls320/400x2100/300	cable carrier chain sls320/400x2100/300	100	100.0
...	...	...	...	...	...
1206	8002213	cable for photocell banner	cable	32	20.0
1098	3020908	pistoke 1-nap. (vaksin)	plug 1-pole (left) pp-h4/1l 3211948	31	20.0
72	0253091	kytkentärima 4	terminal strip 4 stfl-1933604-9 valkoinen	30	20.0
519	2014582	suojavaippaliitin sk 35	protective casing connector sk 35 nro 3026483	26	20.0
1375	M00004882	cable temposonic m12 (8pin) female, angled, ip...	cable	17	0.0

2044 rows × 5 columns

Kuva 11. Nimikedatasarakkeiden vertailu ja samankaltaisuuden analyysi Fuzzy String Matching metodilla

Lopuksi lasketaan jokaiseen samankaltaisuusryhmään kuuluuvan rivimäärän (kuva 12).

```python
# Counting number of rows in each bin
df1 = df.groupby('bin')['Similarity'].count()
df1
```

```
bin
0.0        1
20.0      13
40.0     143
60.0     547
80.0     604
100.0    736
Name: Similarity, dtype: int64
```

Kuva 12. Tarkistettavan datan rivit jaettuna ryhmiin samankaltaisuusprosentin mukaan

Kuva 12 osoittaa, että vain 736:n (vähän yli 1/3 osaa) nimikkeiden 2044:sta nimikenttä on täytetty täysin säännöstöjen mukaisesti, muissa riveissä on enemmän tai vähemmän puutteita. Tämä on vain yhden kentän oikeellisuustarkistusesimerkki, mutta se jo kertoo paljon nimikedatan laadusta.

Esimerkissä käytetty Fuzzy String Matching metodi on vielä yksi tehokas työkalu tekstidatan analyysiin. Se on tekniikka, jota käytetään tunnistamaan tekstielementit, jotka täsmäävät osittain, mutta eivät kokonaan. Pykes (2023) kirjoittaa, että tätä metodia käytetään paljon hakukoneissa. Jos käyttäjä esimerkiksi kirjoittaisi Googleen "Lonto" sanan "Lontoo" sijaan, hakukone Fuzzy String Matching:in avulla tunnistaisi, että haettu sana oli "Lontoo" ja antaisi hakutulokset oikein. Metodin algoritmi mittaa kahden merkkijonojen muokkausetäisyyttä (Edit distance). Muokkausetäisyys määrittää, kuinka lähellä kaksi merkkijonoa ovat toisistaan laskemalla vähimmäismäärä muokkauksia, jotka tarvittaisi yhden merkkijonon muuntamiseen toiseksi. Tässä tapauksessa muokkaus on toimenpide, joka suoritetaan merkkijonolle sen muuttamiseksi toiseksi merkkijonoksi. Muokkaustoiminnot ovat seuraavat: merkin lisäys (Insert), merkin poisto (Delete), kahden vierekkäisen merkin paikkojen vaihto (Switch) ja merkin korvaus toiseen (Replace). Nämä neljä toimintoa mahdollistavat minkä tahansa merkkijonon muokkaamisen. Esimerkiksi "Lonto" ja "Lontoo" sanojen muokkausetäisyys on yksi, koska o-kirjaimen lisääminen sanan loppuun johtaa sanojen täyteen samankaltaisuuteen. Muokkausetäisyyden laskentaan käytetään erilaisia menetelmiä muun muassa Levenshteinin tai Hammingin etäisyyttä. Fuzzy String Matching tekniikan käyttöä Pythonissa mahdollistaa *fuzzywuzzy* kirjastopaketti. Tämä kirjasto käyttää Levenshtein-muokkausetäisyyttä laskeakseen kahden merkkijonon läheisyyttä. Se tarjoaa myös ominaisuuksia merkkijonojen samankaltaisuuden määrittämiseen eri tilanteissa. Kirjaston yleisimmät algoritmit ovat Fuzz Ratio, Fuzz Partial Ratio, Token Set Ratio ja Token Sort Ratio.

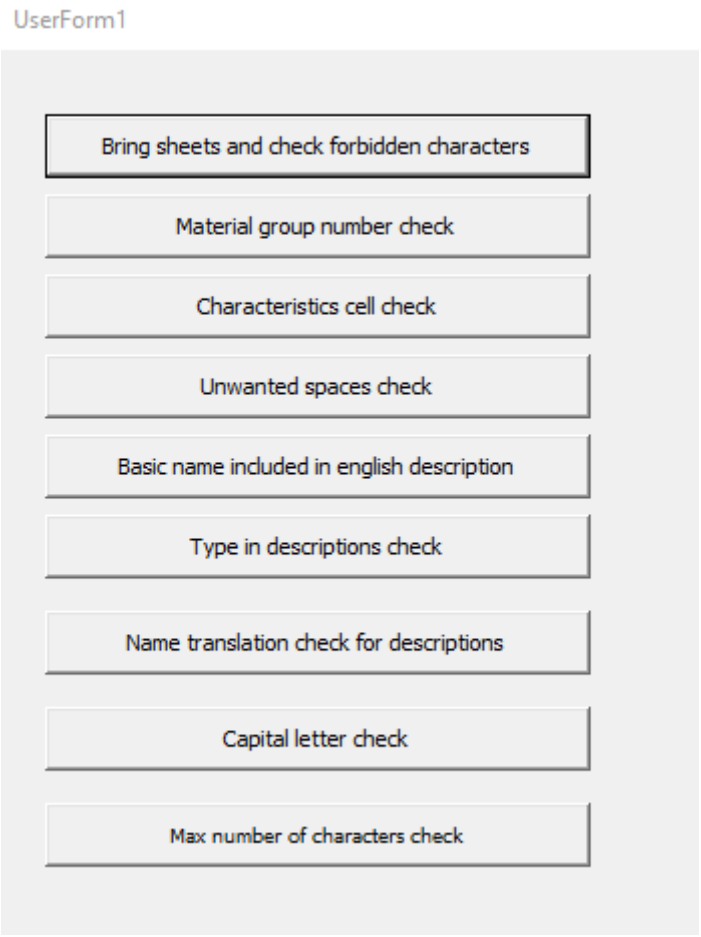
Token Sort Ratio algoritmi, jota käytettiin edellisessä esimerkissä, tokenisoi molemmat syötemerkkijonot, lajittelee tokenit aakkosjärjestykseen ja laskee samankaltaisuuspisteet lajiteltujenmerkkijonoloistojen välissä käyttäen Fuzzy String Matching tekniikkaa. Tämä algoritmi vertailee merkkijonon sisältämät sanat, välittämättä niiden järjestyksestä, mikä sopii tämäntyyppisen nimikedatan tarkistukseen.

3.3.2 Nimikedatan tarkistustyökalun kehitys

Opinnäytetyön seuraava käyttötapaus pohjautui toisen asiakasyritykseen nimikedataan. Kyseessä oli nimikedatan harmonisointiprojekti. Projektin tarkoitus oli yhdistää eri yritysten tytäryhtiöiden nimikkeet ja niihin liittyvät tiedot saman ERP-järjestelmään alle. Niin kuin ensimmäisessä käyttötapauksessa, nimikedatalla on ollut pitkä historia, nimikkeet olivat käyneet useiden järjestelmien ja käsittelijöiden läpi, siksi datan laatu ei ollut paras mahdollinen

tässäkään tapauksessa. Kuitenkin tällä asiakasyrityksellä toisin kuin edellisellä, oli olemassa tarkka datastandardi, joka huomattavasti selkeytti nimikedatan ladun parannustoiminnot. Datan harmonisointimenetelmiin pitkälti kuului olemassa olevan datan vertailu standardiin ja löydettyjen eroavaisuuksien ja puutteiden korjaaminen. Nimikedatan rikastajat saivat yrityksen ERP-järjestelmästä ajetut Excel-tiedostot, jotka sisälsivät nimikedataa tuoteryhmittäin. Tiedostoilla oli tarkkaan määritelty formaatti: Excelin vasemmalla puolella olivat attribuuttisarakkeet, jotka sisälsivät tämänhetkisen nimikedatan. Oikealla puolella olivat tyhjät sarakkeet, jotka täytettiin tarkistetulla harmonisoidulla datalla. Attribuuttisarakkeita oli yli 70, joten koko tiedosto sisälsi yli 150 saraketta. Tällaisen tiedoston täyttäminen ja tarkistus oli todella työläs. Jopa siinä tapauksessa, kun kaikki tiedot löytyivät alkuperäisestä datasta ja olivat kunnossa, yhden nimikkeen tarkistukseen meni noin 20 minuuttia. Usein kaikkia tarvittavia tietoja ei löytynyt tai ne olivat puutteellisia, mikä aiheutti lisäselvittelyjä ja yhteydenottoja asiakasyritykseen tai tuotetoimittajaan. Silloin nimikedatan saaminen kuntoon oli voinut kestää useita tunteja tai jopa päiviä. Koko projektin aikana oli tarkoitus käydä läpi kymmeniä tuhansia nimikkeitä, joten kaikki automatisoidut tai puoliautomatisoidut ratkaisut, joilla olisi voitu helpottaa ja nopeuttaa datan käsittelyprosessia, myös olisivat säästäneet asiakkaalle aikaa sekä kustannuksia.

Rikastustiimin käytössä oli Excel-makroilla tehty työkalu harmonisoidun dataan tarkistukseen. Työkalun avulla oli mahdollista tarkistaa tietyt nimikeattribuutit datastandardimääräyksiä vastaan. Jokaiselle tarkistusoperaatiolle oli tehty oma nappi (kuva 13), joka käynnisti tarkistustoiminnon. Tarkistuksen jälkeen ruudulle ilmestyi ponnahdusikkuna, jossa kerrottiin, menikö tarkistus onnistuneesti, vai oliko datassa puutteita. Puutteellista dataa sisältävät solut maalattiin punaisella värillä.



Kuva 13. Excel-tarkistustyökalun käyttöliittymä

Työkalun käyttö helpotti datan manuaalitarkistuksen huomattavasti. Kuten muiden Excel-työkalujen tapauksessa, tämän ratkaisun etuna oli hyvä saavutettavuus ja kohtuullisen helppo käytettävyys, eikä työkalun implementointi vaatinut ohjelmistointegraatioita. Kuitenkin tässä ratkaisussa olivat myös omat heikot kohtaansa. Koska työkalussa käytettiin Excel-makroja, sen käyttö oli mahdollista vain .xlsm laajennuksella olevalla tiedostolla, mutta tulo- sekä lähtödatan oli oltava .xlsx-muodossa. Tämä ratkaistiin sillä, että työkalu toi aktiivisen Excel-työkirjan lehden .xlsm-tiedostoon ja käsittelyn jälkeen lehti kopioitiin tai tallennettiin perus Excel-muotoon. Käyttäjien toivomuksena olisi ollut mahdollisuus suorittaa tarvittavat tarkistukset samassa alkuperäisessä Excel-tiedostossa. Toinen isompi ongelma oli siinä, että tarkistettavat sarakkeet oli kovakoodattu makroihin niiden Excel-taulukkonumeroilla (esimerkiksi E8) ja pienetkin lähtötiedoston muutokset aiheuttivat väärien sarakkeiden tai sarakevälien tarkistukset, silloin työkalu ei toiminut tarkoituksenmukaisesti.

Tämän opinnäytetyön käyttötapauksena ja nimikedatan harmonisointiprojektin osana oli tarkistustyökalun parannetun version kehittäminen. Uudessa työkaluversiossa nimikeattribuuttien tarkistukset suoritettiin Python skriptin avulla, skriptin ajo käynnistettiin erillisessä .xlsm-tiedostossa olevalla napilla ja tarkistuksen tulokset palautettiin alkuperäiseen Excel-

tiedostoon. Jokaisen tarkistusoperaation tulos tallennettiin omaan sarakkeeseen, tarkistussarakkeet sijoitettiin nimikedatasta vapaaseen tilaan tiedostoon oikealla puolella. Jatkokehitysvaiheessa työkaluun oli kehitetty ja toteutettu VBA-pohjaiset värikoodaukset puutteellista dataa sisältäville soluille. Alla on työkalun kehityksen vaiheet.

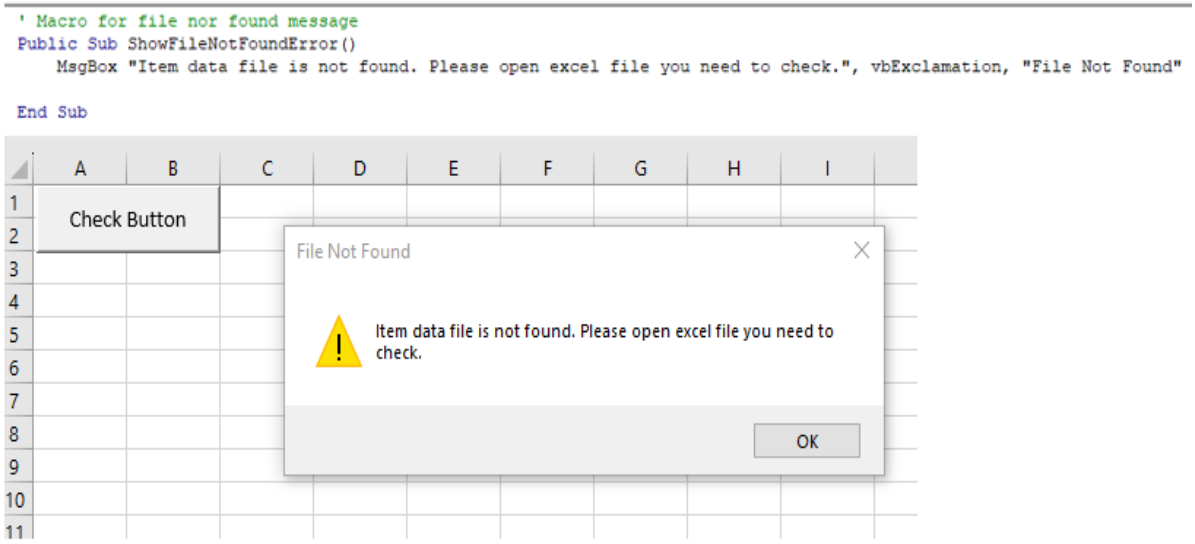
Työkalun käyttöliittymä ja Python skriptin ajo Excelin kautta

.xlsm-tiedostossa olevaan tarkistusnappiin liitettään kuvassa 14 oleva VBA makro.

```
Sub RunPython()

    Dim objShell As Object
    Set objShell = VBA.CreateObject("Wscript.Shell")

    Dim PythonExe As String
      PythonExe = """C:\Users\AppData\Local\anaconda3\python.exe"""

    Dim PythonScript As String
    PythonScript = "C:\Users\PycharmProjects\pythonProject\testl.py"

    Dim Command As String
    Command = PythonExe & " " & PythonScript

    objShell.Run Command

End Sub
```

Kuva 14. VBA makro Python skriptin ajamiseen

Kyseinen makro oli aika yksinkertainen, siinä ilmoitettiin polut python.exe sekä python-koodin sisältävään tiedostoihin, näiden avulla ensin käynnistettiin Pythonia ja siten ajettiin skripti. Tämän makron lisäksi tarkistus Exceliin oli luotu makrot, joiden avulla käyttäjä sai ilmoitusviestejä työkalulta. Jos nimikedatan tarkistus oli päättynyt onnistuneesti, käyttäjä sai ilmoituksen tästä ja tarkistussarakkeet tulostuivat tarkistettavan Excel-tiedoston oikealle puolelle. Jos tarkistettavaa tiedostoa ei löytynyt tai skriptin ajo ei onnistunut jonkun virheen vuoksi, käyttäjä sai virheilmoituksen. Kuvassa 15 on esimerkki ilmoitusviestimakrosta sekä ilmoitusikkunasta. Ilmoitusmakrot aktivoituivat Python skriptillä.



Kuva 15. Ilmoitusviestimakro ja File Not Found viesti-ikkuna

Python skriptin yhdistäminen Exceliin ja datan lataus

Python skripti otti yhteyttä Exceliin *xlwings* moduulia käyttäen. Ohjelma tarkisti kaikki avoimet Excel-tiedostot (siis tiedoston sijainnilla ei ollut väliä) ja latasi dataa tiedostosta, jonka laajennus oli .xlsx (ei .xlsm) ja joka sisälsi tietynnimisen välilehden (*sheetname*-muuttujan mukaan). Jos tarvittavaa tiedostoa tai välilehteä ei löytynyt, funktio *display_message_box_in_excel* aktivoi ilmoitusmakron ja käyttäjä sai kuvassa 15 esitetyn *File Not Found* viestin. Kuva 16 näyttää Exceliin yhdistämisprosessin koodia.

```
import pandas as pd
import xlwings as xw
import re
import os
import string
```

```
# function to check if any excel workbook is open
1 usage
def is_excel_open():
    excel_instances = xw.apps.keys()  # Get a list of all open Excel instances
    for instance in excel_instances:
        app = xw.apps[instance]
        if len(app.books) > 0:  # Check if the instance has any open workbooks
            return True
    return False

# Step 1: finding opened item data Excel file and sheet based on name
if is_excel_open():
    wb1 = None
    opened_workbooks = xw.books
    found = False
    for wb in opened_workbooks:
        # finding open xlsx-file with sheetname Standardization Template
        if wb.name.endswith(".xlsx"):
            for sheet in wb.sheets:

                # finding a sheet with the name Standardization Template
                if sheet.name == sheetname:
                    wb1 = wb
                    found = True
    if found:
        ws = wb1.sheets[sheetname].used_range.value
    if not found:
        print("Workbook is not found!")
        display_message_box_in_excel(check_book_name, not_found_macro)
        exit()

else:
    print("No Excel workbook is open.")
    exit()
```

Kuva 16. Python skriptin yhdistäminen Exceliin tiedoston ja välilehden nimien perusteella

Jos tarvittavat tiedosto ja välilehti löytyivät, välilehden data ladattiin pandas dataframeen jatkokäsittelyä varten. Datan esikäsittelyvaiheessa tyhjät solut täytettiin ”none” arvolla ja

poistettiin ensimmäiset 6 riviä, että otsikkorivi näyttäisi oikein. Nämä vaiheet ovat näytetty kuvassa 17.

```
# Step 2: loading data from excel to be checked to pandas dataframe
df = pd.DataFrame(ws)
df = df.fillna('none')# filling empty cells with 'none' value

# dropping first 6 rows of a table and making row 7 a header row
df = df.iloc[6:].reset_index(drop=True)
df.columns = df.iloc[0]
df = df[1:].reset_index(drop=True)
#copying data to a new dataframe
df1 = df.copy()
```

Kuva 17. Excelin datan lataus pandas dataframeen ja datan esivalmistelu

Tarkistustyökalun toiminnallisuudet

Tarkistustyökalu suoritti tiettyjen attribuuttien tarkistuksen eri kriteereillä. Tämä työkalun versio sisälsi 9 tarkistusoperaatiota. Muutamassa operaatiossa tarkistustiedoston dataa vertailtiin standardidataan. Tätä varten tarkistukseen tarvittavat standardi Excelin välilehdet ladattiin omiin datafameihin (kuva 18).

```
# Step 3: reading data from standard file to pandas dataframe

appendix_file_path = find_file(path, filename_start: 'Appendix')
# reading data from different sheets to different dataframes
df_app= pd.read_excel(appendix_file_path, header = 3, sheet_name='Matl grp specific instructions')
df_app1= pd.read_excel(appendix_file_path, header = 4, sheet_name='Basic Name Translations')
```

Kuva 18. Standarditiedoston datan lataus pandas dataframeen

Kun kaikki tiedot oli ladattu Pythoniin, voitiin toteuttaa työkalun tarkistustoiminnallisuudet.

Ensimmäinen tarkistusoperaatio selvitti, löytyikö harmonisoitujen nimikkeiden materiaaliryhmä standarditiedostosta. Tämä tarkistus oli helppoa suorittaa käyttäen *isin()* metodia. Kuitenkin haasteena tässä tapauksessa oli se, että toisen tiedoston materiaaliryhmät oli luettu pandakseen numeerisena datana ja etunollat oli kadonneet riveiltä, kun toisen tiedoston vastaava sarake oli ladattu Pythoniin tekstityyppisenä. Joten ennen vertailun suorittamista numeerisen sarakkeen data piti muuttaa string-tyyppiseksi ja lisätä siihen etunollat. Tämän jälkeen vertailu onnistui ja vertailutulokset oli lisätty uuteen sarakkeeseen (kuva 19).

```
# Check 1: Checking if material group from harmonized data can be found in Appendix1
# changing column type from int to str
df_app['Material Grp'] = df_app['Material Grp'].astype(str)
# adding heading zeroes
df_app['Material Grp'] = df_app['Material Grp'].apply(lambda x: x.zfill(6))

df1['Mat_Group_Check'] = df1.Material_Group_S.isin(df_app['Material Grp'])
```

Kuva 19. Nimikkeiden materiaaliryhmän vertailu standardidataan

Vastaavasti *isin()* metodia käyttäen tarkistettiin, löytyisivätkö nimikkeiden peruskuvaukset standardidatasta sekä nimikkeiden englanninkielisestä kuvauksesta (kuva 20).

```
#########################################################################################################
# Check 2: Check if description from 'Characteristic_Basic_Name_S' can be found
# in 'Basic Name EN' in Appendix2
df1['Name_Translation_Check'] = df1.Characteristic_Basic_Name_S.isin(df_app1['Basic Name EN'])
#########################################################################################################
# Check 4: checking if Characteristic_Basic_Name from harmonized data can be found
# in Appendix2 column 'Characteristic'
df1['Characteristic_Check'] = df1.Characteristic_Basic_Name_S.isin(df_app.Characteristic)
#########################################################################################################
# Check 5: checking if Characteristic_Basic_Name_S is a part of Description_EN_S column in harmonized data
df1['Basic_Name_Check'] = df1.apply(lambda x: x['Characteristic_Basic_Name_S'] in x['Description_EN_S'], axis=1)
```

Kuva 20. Nimikekuvausten vertailu standardidataan

Toinen tarkistusoperaatioista tarkisti, löytyikö tuotetyyppidata kaikista nimikkeiden kielikuvauksista, jos kuvauskenttä ei ollut tyhjä. Jos tuotetyyppitieto löytyi, tarkistussarakkeeseen tuli True arvo, jos ei löytynyt, sarakkeeseen tuli False arvo + luettelo sarakkeista, joista kyseinen tieto puuttui. Jos sarake, johon kuvaukset vertaillaan oli tyhjä, tästäkin tuli merkintä tarkistussarakkeeseen. Tämän toiminnon toteutus löytyy kuvasta 21.

```
# Check 6: checking if Type_S column text could be found in language description columns
# if they are not empty. If language column is empty, it marked as true in check column,
# if Type_S column is empty, 'Type_S is empty!' mark is added to check column

# changing column type to string and removing trailing zeroes
df1['Type_S'] = df1['Type_S'].astype(str)
df1['Type_S'] = df1['Type_S'].str.replace('.0', '')

# Function to check if data from one column is part of multiple columns
1 usage
def check_text(row, columns_to_check):
    text_to_check = str(row['Type_S'])
    if text_to_check == 'none':
        return 'Type_S column is empty'

    missing_columns = []
    for col in columns_to_check:
        col_value = str(row[col])  # Convert column value to string
        if col_value != 'none' and text_to_check not in col_value:
            missing_columns.append(col)

    if missing_columns:
        return f"False ({', '.join(missing_columns)})"
    else:
        return True

# Columns to check if Tpe_S data is present
columns_to_check = ['Description_EN_S', 'Description_FI_S', 'Description_SV_S',
                    'Description_ET_S', 'Description_DE_S', 'Description_PL_S',
                    'Description_SK_S']
#df1[columns_to_check] = df1[columns_to_check].astype(str)
# Apply the function to add a new column to df1
df1['Type_Check'] = df1.apply(check_text, columns_to_check=columns_to_check, axis=1)
```

Kuva 21. Nimikkeiden tuotetyyppidatan vertailu nimikkeiden kielikuvauksiin

Nämä edellä mainitut toiminnot tarkistivat nimikedatan oikeellisuuden, eli vastasiko nimikedata sisällöllisesti standardimääräyksiä. Seuraavat toiminnot myös tarkistivat datan oikeellisuutta, mutta samalla vaatimustenmukaisuutta (validity), eli niiden avulla tarkistettiin, oliko nimikedata muodollisesti kunnossa ja vastasiko järjestelmän vaatimuksiin. Ensimmäinen näistä toiminnoista tarkisti, sisälsikö nimikedata kiellettyjä merkkejä. Jos kiellettyjä merkkejä ei ollut, tarkistussarakkeeseen tuli OK-merkintä, jos niitä löytyi, tarkistussarakkeeseen tuli kiellettyjä merkkejä sisältävien sarakkeiden luettelo. Kielletyt merkit etsittiin funktion *check_symbols* avulla. Funktion toiminta sekä käyttöesimerkki on esitetty kuvassa 22.

```
# Check 3: checking harmonized columns for presence of forbidden symbols

# Define columns to exclude
columns_to_exclude1 = ['Comments_S', 'Add_Information1_S', 'Add_Information2_S']

# column range excluding Comments_S column
columns_to_check_symbols = (
    check_interval)[check_interval.columns.difference(columns_to_exclude1)]

# symbols to check
symbols_to_check = ["Ø", "#", "'", "*", "°", ";", "»", "?", "!",
                    "\\", "|", "[", "]", "{", "}", "="]

# function to check if any symbol is present in a cell
1 usage
def check_symbols(row):
    forbidden_symbols = []
    for col in columns_to_check_symbols:
        cell_value = str(row[col])
        for symbol in symbols_to_check:
            if symbol in cell_value:
                forbidden_symbols.append(f'"{symbol}", {col}')
    if forbidden_symbols:
        return ', '.join(forbidden_symbols)
    return 'ok'

#  Apply the function to a new column 'Symbol check'

df1['Symbol_Check'] = df1.apply(check_symbols, axis=1)
```

Kuva 22. Kiellettyjen merkkien tarkistus nimikedatasta

Seuraavan tarkistuksen tarkoitus oli varmistaa, että nimikeattribuuttikentät sisälsivät vain isot kirjaimet, jos attribuuttikenttä ei ollut tyhjä. Tämän toiminnallisuuden toteutuksessa vaikeutena oli jälleen kerran numeerinen data. Ohjelma laski pelkästään numeerista dataa sekä numeroiden ja erikoismerkkien yhdistelmiä sisältävät solut pienillä kirjaimilla kirjoitetuksi tekstiksi ja palautti False arvon. Ongelma saatiin ratkaistua käyttämällä *string* -metodin *ascii_uppercase*-vakiota, joka sisältää kaikki englanninkieliset isot kirjaimet A:sta Z:hen merkkijonona. Funktio *check_uppercase* tarkisti, sisälsikö merkkijono vain isot kirjaimet. Funktion *check_columns* avulla tallennettiin kaikki solut, joista löytyy pieniä kirjaimia omaan listaan, jonka tulostettiin tarkistussarakkeeseen. Kuva 23 esittää tämän operaation logiikan.

```
# Check 7: checking columns for upper case excluding Status_S, Case_ID, Comments_S columns

# Function to check if all letters in a string are in upper case
1 usage
def check_uppercase(s):
    return all(letter in string.ascii_uppercase for letter in s)
# Function to check if all columns are in uppercase or not
1 usage
def check_columns(row, columns_to_check_upper):
    missing_columns = []
    for col in columns_to_check_upper:
        cell_value = row[col]
        if (isinstance(cell_value, str) and cell_value.strip() and cell_value.strip() != 'none'
                and not check_uppercase(cell_value)):

            missing_columns.append(col)

    if missing_columns:

        return f"{', '.join(missing_columns)}"
    else:
        return "ok"

# Define columns to exclude
columns_to_exclude2 = ['Status_S', 'Case_ID', 'Comments_S', 'Case_Classification_S',
                       'Material_Sub_Type_S', 'Plants']

# Exclude columns from check interval
columns_to_check_upper = (
    check_interval)[check_interval.columns.difference(columns_to_exclude2)]

df1['Upper_Case_Check'] = df1.apply(lambda row: check_columns(row, columns_to_check_upper), axis=1)
```

Kuva 23. Nimiketekstidatan kirjainten koon tarkistus

Seuraava tarkistus varmisti, että data ei sisältänyt ylimääräisiä välilyöntejä tekstin edessä, takana tai välissä. Välilyönnit etsittiin regex-tekniikkaa käyttäen (kuva 24). Muiden tarkistuksin tapaan, tarkistustuloksesta riippuen tarkistussarakkeeseen palautettiin joko OK- merkinnän tai virhesarakkeiden luettelon.

```
# Check 8: check for extra spaces in harmonized data
# excluding 'Status_S' and 'Comments_S' columns
# Function to check for heading, trailing and extra spaces in a string
1 usage
def check_spaces(s):
    if isinstance(s, str):
        return bool(re.search( pattern: r'^\s+|\s+$|\s{2,}', s))
    return False

# Define columns to exclude
columns_to_exclude3 = ['Status_S', 'Comments_S']

# Columns to check for spaces
columns_to_check_spaces = (
    check_interval[check_interval.columns.difference(columns_to_exclude3)].columns.tolist())

# Check columns for spaces
spaces_check = df1[columns_to_check_spaces].applymap(check_spaces)

# Create a new column indicating presence of spaces
df1['Space_Check'] = spaces_check.apply(lambda row: ', '.join([col for col,
space in zip(columns_to_check_spaces, row) if space]),
axis=1).apply(lambda x: 'ok' if x == '' else x)
```

Kuva 24. Ylimääräistään välilyöntien tarkistus nimikedatasta

Viimeisessä tarkistusoperaatiossa varmistettiin, että kielikuvauskenttien sisältö ei olisi ylittänyt 40 merkkiä ja valmistajanimikekoodin pituus oli 30 merkkiä. Kenttien pituuksien vertailu raja-arvoon toteutettiin *check_length* funktiolla. Ensin kielikuvausdata ja valmistajakoodit tarkistettiin erikseen ja tarkistustulokset lisättiin omiin sarakkeisiin, sitten sarakkeet yhdistettiin yhdeksi tarkistussarakkeeksi niin, että jos kummassakin sarakkeessa olisi ollut OK merkintä, se olisi siirtynyt tulossarakkeeseen, jos jommassakummassa tai kummassakin sarakkeessa oli merkintöjä pituuden ylittämisestä, ne merkinnät olisivat siirtyneet uuteen sarakkeeseen. Lopulta ylimääräiset apusarakkeet poistettiin. Kuva 25 esittää tämän toiminnallisuuden.

```
# Check 9: checking for columns' data length, 'Description_EN_S' - 'Description_SK_S' max 40 characters
# 'Manufacturers_Item_S' max 30 characters
# Function to check if the length of value is not longer than limit value
2 usages
def check_length(s, limit):
    return len(str(s).strip()) <= limit

# Range of columns to check the length
columns_to_check1 = df1.loc[:, 'Description_EN_S': 'Description_SK_S'].columns.tolist()
columns_to_check2 = df1.loc[:, 'Manufacturers_Item_S': 'Manufacturers_Item_S'].columns.tolist()
# column data length limits
limit1 = 40
limit2 = 30
# Check columns for length
length_check1 = df1[columns_to_check1].applymap(lambda x: check_length(x, limit1))
length_check2 = df1[columns_to_check2].applymap(lambda x: check_length(x, limit2))

# Create a new column indicating if length is within the limit
df1['Check1'] = length_check1.apply(lambda row: 'ok' if row.all()
else ', '.join([col for col, check in zip(columns_to_check1, row) if not check]), axis=1)
df1['Check2'] = length_check2.apply(lambda row: 'ok' if row.all()
else ', '.join([col for col, check in zip(columns_to_check2, row) if not check]), axis=1)

# Function to merge values from two check columns into one
1 usage
def merge_values(row):
    if row['Check1'] == 'ok' and row['Check2'] == 'ok':
        return 'ok'
    else:
        return ', '.join(filter(lambda x: x != 'ok', [row['Check1'], row['Check2']]))

# Apply the function to create the 'Length_Check' column
df1['Length_Check'] = df1.apply(merge_values, axis=1)

# deleting auxiliary columns
df1.drop(['Check1', 'Check2'], axis=1, inplace=True)
```

Kuva 25. Nimikedatakenttien sisällön pituuden tarkistus

Kun kaikki tarkistukset olivat suoritettu, nimikekoodit sekä tarkistussarakkeet tallennettiin uuteen dataframeen (kuva 26). Tämän dataframen sisältö lisättiin tarkistettavan Excel-tiedostoon.

```
# Step 5: when all checks are completed, saving check columns to a new dataframe df2
df2 = df1[['Material_Number_S', 'Symbol_Check', 'Upper_Case_Check', 'Space_Check',
           'Length_Check', 'Mat_Group_Check', 'Type_Check', 'Basic_Name_Check',
           'Characteristic_Check', 'Name_Translation_Check']]
```

Kuva 26. Tarkistussarakkeiden tallennus uuteen dataframeen

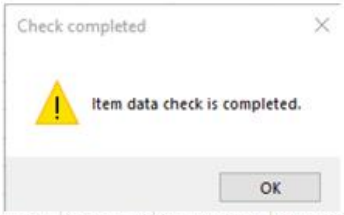
Tarkistustulosten tallentaminen Exceliin

Lopulta ohjelmaa uudestaan otti yhteyttä tarkistettavaan Exceliin samaan välilehteen, josta lähtödata oli saatu ja lisäsi dataframen df2 dataa ensimmäiseen tyhjään sarakkeeseen riville 7. Tällä tavalla ohjelma käsitteli Excelin dataa, mutta ei Excel-tiedostoa itseään, eikä rikkonut alkuperäisen Excelin monimutkaista rakennetta ja muotoilua. Tarkistusdatan tallennus Excelin on esitetty kuvassa 27.

```
    # Step 6: connecting to checked excel and saving check columns to it

    # Get a list of all opened Excel workbooks
    wb2 = None
    opened_workbooks = xw.books

    # Finding files with extension .xlsx with sheetname Standardization Template
    # among opened workbooks
    for wb in opened_workbooks:
        if wb.name.endswith(".xlsx"):
            for sheet in wb.sheets:
                if sheet.name == sheetname:
                    wb2 = wb
    # Check if the target workbook is found
    if wb2 is not None:
        # Access a specific sheet in the workbook
        ws1 = wb2.sheets[sheetname]

        # Adding check columns to first empty column to row 7 in checked excel
        excel_column_name =(
                ws1.used_range[-1].offset(column_offset=1).address.split("$")[1] + str(7))
        ws1.range(excel_column_name).options(index=False).value = df2
        # Save the workbook
        wb2.save()
        # Print a message indicating success
        print(f"Data added to 'Standardization Template' sheet in workbook: {wb1.name}")
    else:
        print(f"No workbook with the name '{wb1.name}' found.")

    # showing check complete macro in excel
    display_message_box_in_excel(check_book_name, check_ok_macro)

# trying to catch any errors stopping the code
# in case of error, user will get notification in excel
except BaseException as e:
    print(f"Error occurred: {e}")
    display_message_box_in_excel(check_book_name, error_macro)
```

Kuva 27. Tarkistusdatan tallennus Exceliin

Jos tallennus on onnistunut, käyttäjä sai viestin, että tarkistus on päättynyt. Jos ohjelma kaatui jonkun virheen vuoksi, koodi aktivoi vastaavan VBA-makron ja käyttäjä sai virheilmoituksen (*display_message_box_in_excel* funktio). Kuva 28 esittää tarkistussarakkeet sekä onnistuneen tarkistuksen ilmoitusikkunan.



Material_	Symbol_Check	Mat_Group_Check	Characteristic_Check	Basic_Name_Chec	Type_Check	Upper_Case_Check	Space_Check	Length_Check	Name_Translation_Ch
193649	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193650	OK	TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
193651	OK	TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
193652	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193653	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193654	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193655	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193681	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193689	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193706	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
193729	OK	TRUE	TRUE	TRUE	Type_S column is emp	not ok (Description_EN_S, Des	ok	ok	TRUE
193768	OK	TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
194239	OK	FALSE	FALSE	TRUE	Type_S column is emp	ok	Equipment_Suppli	ok	FALSE
194324	OK	TRUE	FALSE	FALSE	Type_S column is emp	ok	Equipment_Suppli	ok	FALSE
402446	OK	TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
402761	OK	TRUE	FALSE	FALSE	Type_S column is emp	ok	ok	ok	FALSE
403571	OK	TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
403661	OK	TRUE	FALSE	TRUE	False (Description_EN	ok	ok	ok	FALSE
403677	OK	TRUE	FALSE	TRUE	TRUE	ok	ok	ok	TRUE
403683	OK	TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
403717	OK	TRUE	FALSE	TRUE	TRUE	ok	ok	ok	FALSE
404084	OK	TRUE	FALSE	TRUE	TRUE	ok	ok	ok	FALSE
404274	OK	TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE

Kuva 28. Exceliin tulostetut tarkistussarakkeet

Tarkistettavien datasolujen väritys

Työkalun jatkokehitysvaiheessa nimikerikastajien töiden helpottamiseksi .xlsm- tarkistustiedostoon oli lisätty väritysmakrot. Tarkistustoimintojen tapaan väritystoiminnot olisi voitu suorittaa Python koodin avulla, mutta siinä tapauksessa Excelin rakenne ja muotoilu olisi mahdollisesti muuttunut, ja asiakasvaatimusten mukaan näin ei olisi saanut tapahtua. Ohjelmointilogiikan kannalta tarkistussarakkeet voitaisiin jakaa kahteen ryhmään: toisessa niistä sarakkeet sisälsivät joko True tai False arvot (kuten materiaaliryhmän olemassaolon tarkistus standardidatasta), toisen ryhmän tarkistussarakkeissa virhesoluissa viitattiin yhteen tai useampaan virheellistä dataa sisältävään sarakkeeseen (kuten kiellettyjen merkkien etsintä). Ensimmäiseen ryhmään kuuluville tapauksille oli tehty funktio, jonka avulla False arvot sisältävät solut tarkistussarakkeessa sekä vastaavan sarakkeen samassa rivissä olevat solut maalattiin tietyllä värillä. Funktio *ColorCells* on esitetty kuvassa 29. Funktion argumentteina käytetiin tarkistussarakkeen otsikko, kohdesarakkeen otsikko, tietty ehto (solut väritettiin, jos solun arvo ei ollut ”True” tai ”ok”) ja värikoodi RGB-muodossa.

```
Function ColorCells(ByVal ws As Worksheet, ByVal checkColumnHeader As String, _
ByVal targetColumnHeader As String, ByVal condition As String, ByVal colorRGB As Long) As Long
    Dim lastRow As Long
    Dim checkColumnCol As Long
    Dim targetColumnCol As Long
    Dim currentRow As Long

    ' Find the last row with data in column A
    lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row

    ' Find the column number of the check column header
    checkColumnCol = 0
    For i = 1 To ws.Cells(7, ws.Columns.Count).End(xlToLeft).Column
        If ws.Cells(7, i).Value = checkColumnHeader Then
            checkColumnCol = i
            Exit For
        End If
    Next i

    ' If check column header not found, exit function
    If checkColumnCol = 0 Then
        MsgBox "Column '" & checkColumnHeader & "' not found!"
        ColorCells = 1 ' Return error code
        Exit Function
    End If

    ' Find the column number of the target column header
    targetColumnCol = 0
    For i = 1 To ws.Cells(7, ws.Columns.Count).End(xlToLeft).Column
        If ws.Cells(7, i).Value = targetColumnHeader Then
            targetColumnCol = i
            Exit For
        End If
    Next i

    ' If target column header not found, exit function
    If targetColumnCol = 0 Then
        MsgBox "Column '" & targetColumnHeader & "' not found!"
        ColorCells = 1 ' Return error code
        Exit Function
    End If

    ' Loop through each row from row 8 to the last row
    For currentRow = 8 To lastRow
        If ws.Cells(currentRow, checkColumnCol).Value <> condition Then
            ' If condition is met, color cells
            ' Color check column cell
            ws.Cells(currentRow, checkColumnCol).Interior.Color = colorRGB
            ' Color corresponding cell in target column
            ws.Cells(currentRow, targetColumnCol).Interior.Color = colorRGB
        End If
    Next currentRow

    ColorCells = 0 ' Return success code
End Function
```

Kuva 29. Funktio *ColorCells* virhedatasolujen värittämiseen

Toisen ryhmän tapauksessa tarkistussarakkeesta piti löytää virhedata sisältävien sarakkeiden otsikot ja värittää samalla rivillä olevat solut niissäkin tarkistussarakkeen virhesolujen lisäksi. Tätä varten oli tehty funktio *MultipleColorCells*. Funktion alkuosa on täysin sama kuin *ColorCells* funktiossa, siinä Excelistä etsitään tietynniminen tarkistussarake ja sieltä solut, joiden sisältö ei vastannut ”True” tai ”ok”-arvoa. Erona edellisestä funktiosta oli se, että tässä funktiossa käytettiin *headersArray*-muuttuja, joka sisälsi kaikki mahdolliset sarakkeiden otsikot listan muodossa. Sen jälkeen, kun virhesolu on löytynyt, funktio kävi solun

sisällön läpi ja keräsi sieltä kohdesarakkeiden otsikot toiseen listaan. Seuraavaksi for-luupin avulla funktio iteroi tämän otsikkolistan läpi ja maalasi solut tarkistussarakkeessa sekä yhdessä tai useammissa kohdesarakkeissa tietyllä värillä. Funktion *MultipleColorCells* loppuosan koodi on esitetty kuvassa 30.

```
    ' List of headers
    headersArray = Split("Description_EN,Description_FI,Description_SV,Description_ET,Description_DE,Description_PL, _
    Description_SK,Manufacturers_Item,Type,ID_S,System_S,Material_Number_S,Status_S,Comments_S,Case_Classification_S, _
    Material_Type_S, Material_Sub_Type_S,Base_Unit_of_Measure_S,Material_Group_S,Industry_Standard_Description_PUMA_ID_S, _
    Indicator_In_Bulk_Liquid_S,Commodity_Code_S,1st_Customs_Unit_S,2nd_Customs_Unit_S,Description_EN_S,Description_FI_S, _
    Description_SV_S,Description_ET_S,Description_DE_S, Description_PL_S,Description_SK_S,Characteristic_Basic_Name_S, _
    Manufacturer_S,Manufacturers_Item_S,Type_S,Type2_Other_Product_Code_S,Measure1_Value1_S,Measure2_Value2_S, _
    Measure3_Value3_S,Measure4_Value4_S,Measure5_Value5_S,Material_S,Standard_S,Metsä_Drawing_S,Equipment_Supplier_S, _
    Equipment_Suppliers_ID_S,SSG_Number_S,InternationalArticleNumber_EAN_UPC,Main_EAN,EAN_category_For_EAN_in_Base_UoM, _
    Add_Information1_S,Add_Information2_S,Plants,No_of_plants", ",")

    ' Loop through each row from row 8 to the last row
    For currentRow = 8 To lastRow
        If ws.Cells(currentRow, checkColumnCol).Value <> condition Then
            ' Get the value of the cell in the check column
            cellValue = ws.Cells(currentRow, checkColumnCol).Value
            ' Split the cell value into an array based on commas
            headersInCell = Split(cellValue, ",")

            ' Reset the array of matched headers for each row
            Erase matchedHeaders
            numMatches = 0
            ' Loop through each cell in the check column
            For Each cellHeader In headersInCell
                ' Trim the header to remove leading and trailing spaces
                header = Trim(cellHeader)

                ' Check if the header is found in the list of headers
                matchIndex = Application.Match(header, headersArray, 0)
                If Not IsError(matchIndex) Then
                    ' If found, add the matched header to the array
                    numMatches = numMatches + 1
                    ReDim Preserve matchedHeaders(1 To numMatches)
                    matchedHeaders(numMatches) = header
                End If
            Next cellHeader

            ' Loop through the matched headers in the current row and color corresponding cells
            For i = 1 To numMatches
                If Not IsEmpty(matchedHeaders(i)) Then
                    targetColumnCol = Application.Match(matchedHeaders(i), ws.Rows(7), 0)
                    ' If target header found, color the corresponding cell
                    If Not IsError(targetColumnCol) Then
                        ' Color the check column cell
                        ws.Cells(currentRow, checkColumnCol).Interior.Color = colorRGB
                        ' Color the corresponding cell
                        ws.Cells(currentRow, targetColumnCol).Interior.Color = colorRGB
                    End If
                End If
            Next i
        End If
    Next currentRow

    MultipleColorCells = 0 ' Return success code
End Function
```

Kuva 30. Funktio *MultipleColorCells* useiden sarakkeiden virhedatasolujen värittämiseen

Tämän jälkeen .xlsm-tiedostoon oli lisätty väritysnapit ja makrot, jotka sisälsivät väritysfunktioiden kutsut. Alussa kaikki väritystoiminnot käynnistettiin samalla napilla ja kuvassa 31 esitetyllä makrolla. Makro etsi avatuista .xlsx-tiedostoista se, jolla oli tietynniminen välilehti ja ajoi funktiokutsut sisältävää koodia, jos sopiva tiedosto ja välilehti löytyivät.

```
Sub ColorCellsInOpenWorkbook()
    Dim wb As Workbook
    Dim ws As Worksheet
    'Dim filteredHeaders() As String ' Define an array to hold the filtered headers


    ' Loop through all open workbooks
    For Each wb In Workbooks
        ' Check if the workbook name ends with ".xlsx"
        If LCase(Right(wb.Name, 5)) = ".xlsx" Then
            ' If the naming convention matches, set the worksheet
            Set ws = wb.Sheets("Standardization Template")
            If Not ws Is Nothing Then
                ' Call the ColorCells function with the parameters for your specific columns and conditions
                ColorCells ws, "Mat_Group_Check", "Material_Group_S", "True", RGB(255, 128, 0) ' Red
                ColorCells ws, "Type_Check", "Type_S", "True", RGB(255, 0, 0)
                ColorCells ws, "Type_S_in_Legacy_Data", "Type_S", "True", RGB(255, 0, 0)
                MultipleColorCells ws, "Type_S_in_Legacy_Data", "True", RGB(255, 0, 0)
                ColorCells ws, "Basic_Name_Check", "Characteristic_Basic_Name_S", "True", RGB(255, 0, 255)
                ColorCells ws, "Characteristic_Check", "Characteristic_Basic_Name_S", "True", RGB(255, 153, 153)
                ColorCells ws, "Name_Translation_Check", "Characteristic_Basic_Name_S", "True", RGB(178, 102, 255)

                MultipleColorCells ws, "Symbol_Check", "ok", RGB(226, 168, 8)
                MultipleColorCells ws, "Upper_Case_Check", "ok", RGB(255, 255, 102)
                MultipleColorCells ws, "Space_Check", "ok", RGB(204, 204, 0)
                MultipleColorCells ws, "Length_Check", "ok", RGB(102, 178, 255)


            End If
            Exit Sub ' Exit the loop after finding and processing the workbook
        End If
    Next wb

    ' If no matching workbook is found, display a message
    MsgBox "No open workbook matching the specified criteria found!"
End Sub
```

Kuva 31. VBA-makro, jolla käynnistetiin väritysfunktiot

Kuitenkin väritystoiminnon testausvaiheessa huomattiin, että sama Excel-tiedoston solu mahdollisesti sisälsi useita virheitä, ja vaikka värityksessä käytettiin eri värit eri virheille, koodin ajon jälkeen solu sai värinsä viimeisen funktiokutsun mukaan. Tämä olisi hankaloittanut solun virhetyypin selvittämistä, joten jokaiselle tarkistusoperaatiolle oli tehty oma nappi ja siihen linkitetty oma makro, joka sisälsi yhden funktiokutsun yhtä tarkistussaraketta kohti. Kuvassa 32 on esimerkit erillisistä makroista.

```
Sub Name_Translation_Check()
    Dim wb As Workbook
    Dim ws As Worksheet

    ' Loop through all open workbooks
    For Each wb In Workbooks

        If LCase(Right(wb.Name, 5)) = ".xlsx" Then
            ' If the naming convention matches, set the worksheet
            Set ws = wb.Sheets("Standardization Template")
            If Not ws Is Nothing Then
                ' Call the ColorCells function with the parameters for your specific columns and conditions
                ColorCells ws, "Name_Translation_Check", "Characteristic_Basic_Name_S", "True", RGB(178, 102, 255)

            End If
            Exit Sub ' Exit the loop after finding and processing the workbook
        End If
    Next wb

    ' If no matching workbook is found, display a message
    MsgBox "No open workbook matching the specified criteria found!"
End Sub
```

```
Sub Symbol_Check()
    Dim wb As Workbook
    Dim ws As Worksheet

    ' Loop through all open workbooks
    For Each wb In Workbooks

        If LCase(Right(wb.Name, 5)) = ".xlsx" Then
            ' If the naming convention matches, set the worksheet
            Set ws = wb.Sheets("Standardization Template")
            If Not ws Is Nothing Then
                MultipleColorCells ws, "Symbol_Check", "ok", RGB(226, 168, 8)

            End If
            Exit Sub ' Exit the loop after finding and processing the workbook
        End If
    Next wb

    ' If no matching workbook is found, display a message
    MsgBox "No open workbook matching the specified criteria found!"
End Sub
```

Kuva 32. Tarkistustoimintokohtaiset makrot väritysfunktioiden kutsuun

Esimerkki väritetyistä tarkistussarakkeiden soluista löytyy kuvasta 33.

Mat_Group_Check	Characteristic_Check	Basic_Name_Che	Type_Check	Upper_Case_Check	Space_Check	Length_Check	Name_Translation_Ch
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	not ok (Description_EN_S, Des	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
FALSE	FALSE	TRUE	Type_S column is emp	ok	Equipment_Supplı	ok	FALSE
TRUE	FALSE	FALSE	Type_S column is emp	ok	Equipment_Supplı	ok	FALSE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	FALSE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	False (Description_EN	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	TRUE	ok	ok	ok	TRUE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	TRUE	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	TRUE	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	FALSE	TRUE	Type_S column is emp	ok	ok	ok	FALSE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE
TRUE	TRUE	TRUE	Type_S column is emp	ok	ok	ok	TRUE

Kuva 33. Väritetyt virhesolut tarkistussarakkeissa

Charachteristic_Check saraketta vastaavan kohdesarakkeen väritykset esittää kuva 34.

Characteristic_Basic_Name_S	Material	Symbol_Check	Mat_Group_Check	Characteristic_Check
	190856	OK	TRUE	FALSE
SHAFT SEAL	190857	OK	TRUE	TRUE
	190926	OK	TRUE	FALSE
SHAFT SEAL	193649	OK	TRUE	TRUE
	193650	OK	TRUE	FALSE
	193651	OK	TRUE	FALSE
O-RING	193652	OK	TRUE	TRUE
SEALING RING	193653	OK	TRUE	TRUE
FLANGE GASKET	193654	OK	TRUE	TRUE
O-RING	193655	OK	TRUE	TRUE
SHAFT SEAL	193681	OK	TRUE	TRUE
O-RING	193689	OK	TRUE	TRUE
SHAFT SEAL	193706	OK	TRUE	TRUE
O-RING	193729	OK	TRUE	TRUE
SHAFT SEAL	193768	OK	TRUE	TRUE
	194239	OK	FALSE	FALSE
	194324	OK	TRUE	FALSE
OIL SEAL	402446	OK	TRUE	FALSE
	402761	OK	TRUE	FALSE
OIL SEAL	403571	OK	TRUE	FALSE
OIL SEAL	403661	OK	TRUE	FALSE
SEAL KIT	403677	OK	TRUE	FALSE
OIL SEAL	403683	OK	TRUE	FALSE
OIL SEAL	403717	OK	TRUE	FALSE
DUST SEAL	404084	OK	TRUE	FALSE
OIL SEAL	404274	OK	TRUE	FALSE
SHAFT SEAL	404709	OK	TRUE	TRUE
SHAFT SEAL	404710	OK	TRUE	TRUE
OIL SEAL	404978	OK	TRUE	FALSE
OIL SEAL	404995	OK	TRUE	FALSE
OIL SEAL	404996	OK	TRUE	FALSE

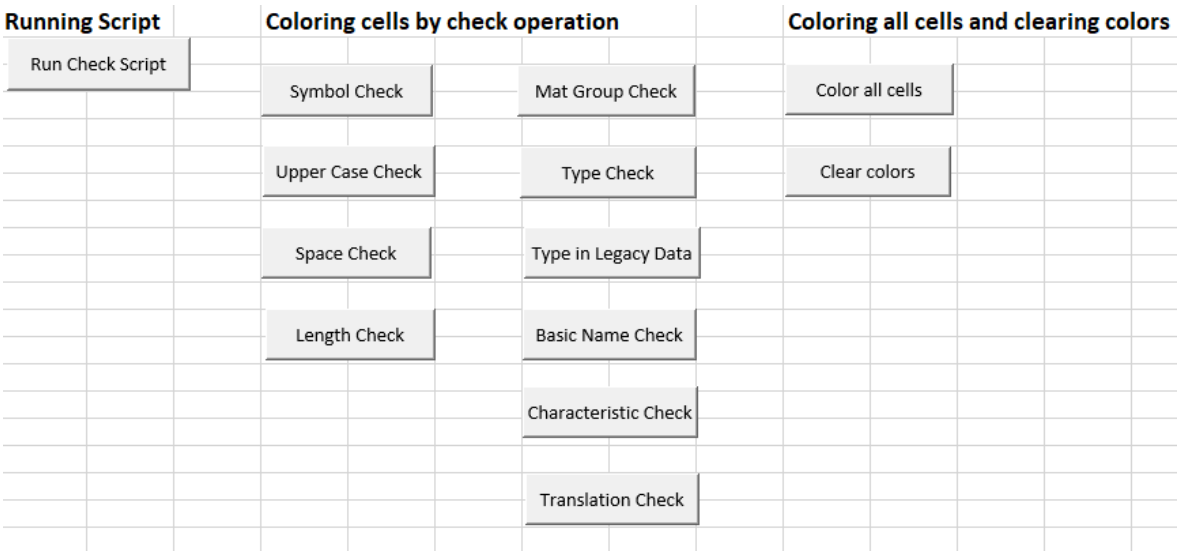
Kuva 34. Kohdesarakkeen väritys tarkistussarakkeen dataan mukaan

Lopuksi tarkistustyökaluun oli lisätty nappi ja makro, joiden avulla kaikki värikoodaukset rivistä 8 alkaen oli mahdollista poistaa tarkistuksen ja korjausten jälkeen (kuva 35). Solujen väritykset sekä tarkistussarakkeet poistettiin tarkistetusta tiedostosta ennen palautusta asiakkaalle.

```
Sub ClearCellColorings()
    Dim wb As Workbook
    Dim ws As Worksheet
    Dim lastRow As Long
    Dim rng As Range

    ' Iterate through all open workbooks
    For Each wb In Application.Workbooks
        ' Check if the workbook name  and ends with ".xlsx"
        If LCase(Right(wb.Name, 5)) = ".xlsx" Then
            ' Check if the sheet exists in the workbook
            On Error Resume Next
            Set ws = wb.Sheets("Standardization Template")
            On Error GoTo 0

            If Not ws Is Nothing Then
                ' Find the last row with data
                lastRow = ws.Cells(ws.Rows.Count, "A").End(xlUp).Row

                ' Check if there is data starting from row 8
                If lastRow >= 8 Then
                    ' Define the range to clear the colorings
                    Set rng = ws.Range("A8:XFD" & lastRow)

                    ' Clear cell colorings
                    rng.Interior.ColorIndex = xlNone
                End If
            End If
        End If
    Next wb
End Sub
```

Kuva 35. Virhesolujen värien poistamismakro

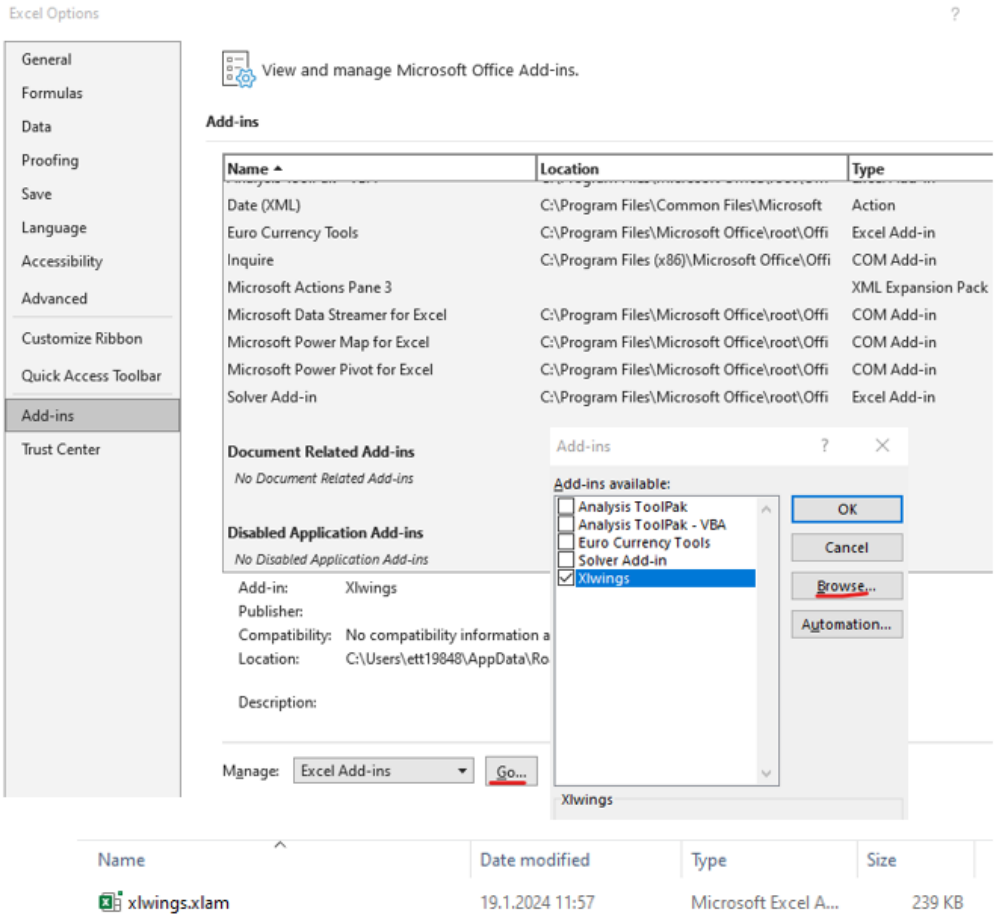
Työkalun lopullinen käyttöliittymä on esitetty kuvassa 36. Kaikkien virhesolujen väritystä käynnistettävää nappia päätettiin säilyttää työkalussa mahdollisia pika- tai lopputarkistuksia varten.



Kuva 36. Tarkistustyökalun käyttöliittymä

Työkalun implementointi ja hyödyt

Opinnäytetyön kirjoitushetkellä tarkistustyökalu oli otettu käyttöön toimeksiantajan Suomen toimipisteen testikäyttäjien tietokoneilla. Ratkaisu oli todettu toimivaksi ja huomattavasti tarkistustoimintoja helpottavaksi. Useat nimikedataa sisältävät tiedostot oli jo käyty läpi työkalun avulla. Seuraavaksi tarkistustyökalun käyttöä oli opastettu ulkomailla toimiville rikastustiimeille laajennettua implementointia varten. Työkalun käyttöönotto on vaatinut seuraavat asennukset käyttäjäkoneella: ensin oli asennettava Python-ohjelmisto käyttäen *python.exe* tiedostoa, tämän jälkeen koneelle piti asentaa Pythonin *xlwings*, *pandas* ja *openpyxl* kirjastot. Asennukset tehtiin komentokehotteen kautta *pip*-asennusohjelman avulla (esimerkiksi *pip install xlwings*). Seuraavaksi xlwings moduuli piti lisätä Exceliin. Tämä tehtiin Excel-tiedoston kohdasta File > Options > Add-Ins, ikkunan alareunasta "Manage"-kohdan vieressä valittiin Excel Add-ins, sitten painettiin Go >Browse ja siirrettiin kansioon, johon xlwings-apuohjelma asennettiin (yleensä Pythonin Lib\site-packages\xlwings\addin-kansio), siinä valittiin apuohjelmatiedosto (xlwings.xlam) ja painettiin OK. Lisäksi piti varmistaa, että Xlwings add-in oli ruksattuna Add-ins luettelossa. Xlwings apuohjelman lisäämisvaiheet ovat esitetty kuvassa 37.



Kuva 37. Xlwings apuohjelman lisääminen Exceliin

Kun nämä asennukset oli tehty, Python-skripti ja tarkistustiedosto tallennetiin käyttäjän tietokoneelle, muutettiin kovakoodatut tiedostopolut sekä skriptissä että *RunPython* makrossa ja työkalu oli käyttövalmis.

Työkalun käyttöönoton varhaisessa vaiheessa käyttäjiltä tuli positiivista palautetta, työkalua alettiin käyttämään ennen solujen väritystoiminnon kehittämistä, koska jo silloin se oli varmempi ja selkeämpi kuin edellinen puhtaasti VBA-makroilla tehty ratkaisu. Väritystoiminto lisäsi työkaluun käyttäjäystävällisyyttä ja helpotti monien sarakkeiden tarkastelua sekä pysty- että vaakasuunnassa. Työn helpottamisen ja nopeuttamisen lisäksi tarkistustyökalun avulla pystyttiin löytämään silmämääräisesti vaikeasti havaittavissa olevat virheet kuten ylimääräiset välilyönnit. Toisin sanoen työkalu tehosti datanladun parantamisprosessia ja sillä säästi asiakkaan ajan ja kustannukset. Suhteellisen pienillä muokkauksilla työkalua voisi mahdollisesti käyttää jatkossa muiden yritysasiakkaiden nimikedatan tarkistuksessa, varsinkin silloin, kun yrityksellä on tarkka datastandardi ja vertailu- / tarkistuskriteerit voi määrittää selkeästi.

4 Yhteenveto ja pohdinta

Tässä opinnäytetyössä nimikedata ja siihen liittyvät haasteet käsiteltiin master data hallinnan näkökulmasta, koska nimikkeet ovat merkitsevä osa yrityksen master dataa. Datan laadun ulottuvuuksien kannalta nimikedatan laadun tärkeimmät kriteerit ovat datan oikeellisuus, ainutlaatuisuus, kattavuus, yhteneväisyys, vaatimustenmukaisuus ja ajantasaisuus. Eri yrityksillä voi olla painepisteet eri datan ladun ulottuvuuksissa, kuitenkaan nimikedataa ei voi pitää laadukkaana, jos nämä kriteerit eivät täyty. Huononlaatuinen nimikedata on nimikedatahallinnan suuri haaste. Jos nimiketiedot eivät ole oikeat tai puuttuvat kokonaan, nimikekannassa on duplikaatteja tai käytöstä poistuneita nimikkeitä, nimikedata ei voi palvella yrityksen liiketoimintaa tarkoituksenmukaisesti ja yritys ei voi välttyä virheistä, ajanhukasta ja ylimääräisistä kustannuksista. Suurin nimikehallinnan haaste on nimikedatan jatkuva kasvu nykyisessä digitaaliympäristössä. Monissa yrityksissä vuosien saatossa on ehtinyt kertyä paljon hallitsemattomasti luotua nimikedata, jossa on lukuisia laatuvirheitä. Nämä virheet hankaloittavat tai jopa estävät yrityksen sujuvaa toimintaa. Isojen datamassojen laatua voidaan parantaa käynnistämällä yrityksessä niin sanottu nimikedatan suursiivousprojekti. Projektin toimenpiteet voivat erota yritysten tarpeiden mukaan, kuitenkin ennen varsinaisten laadunparannustoimintojen alkamista on päätettävä, mikä on hyvänlaatuinen data juuri kyseiselle yritykselle ja luoda selkeän datastandardin, joka vastaisi organisaation vaatimuksiin. Vasta näiden vaiheiden jälkeen voi ryhtyä toimeen. Suuret nimikedatan laadun huoltoprojektit yleensä käynnistetään yritysten toiminnan solmupisteissä, kuten esimerkiksi uuden tietojärjestelmän käyttöönotto tai datan migraatio järjestelmien välissä. Yritysten koosta ja tarpeista riippuen "suurisiivoukset" tehdään noin 5–10 vuoden välein. Kuitenkin pelkästään datan laadun kertaluontoiset huoltotoimenpiteet eivät riitä. Datan hyvää laatua voi varmistaa vain jatkuvilla ylläpitotoiminnoilla. Nimikkeiden määrän kasvun on oltava hallittua, on tärkeitä, että nimikkeet luodaan, päivitetään ja poistetaan käytöstä yhtenäisten sääntöjen ja prosessien mukaan. Nämä säännöt ja prosessit pitäisi olla tiedossa kaikilla nimikedataa käsittelevillä sidosryhmillä. Kun nimikedata huolletaan jatkuvasti ja pelisäännöt ovat kaikille selvät, yrityksen datan laatu taso nousee huomattavasti, laatuvirheiden määrä pienenee ja organisaatio voi keskittyä omaan ydintoimintaan sen sijaan että jatkuvasti kuluttaisi aika ja resurssit hätäkorjaustoimenpiteisiin.

Opinnäytetyön käytännön osuuden tavoitteena oli löytää automatisoidut ratkaisut näiden nimikedataan liittyviin ongelmatapausten havaitsemiseen: nimikkeiden kaksoiskappaleet ja puuteellinen nimikedata. Nimikkeiden kaksoiskappaleet (duplikaatit) etsittiin nimikedatasta sekä perus Excel hakutoimintoja (Conditional formating →Duplcate values) käyttäen että koneoppimismallin ja Python koodauksen avulla. Molemmat ratkaisut toimivat hyvin täydellisten duplikaattien haussa suhteellisen pienissä datasetteissä. Kummankin ratkaisun avulla

yli 2000 nimikettä sisältävistä datasetistä oli löytynyt noin sata duplikaattiparia. Excel-pohjaisen ratkaisun etuna on helppo saavutettavuus ja vapaus koodaamisesta. Koneoppimismallin etuja ovat skaalautuvuus ja duplikaattihakutulosten tulostusmahdollisuus erilliseen tiedostoon. Kuitenkin jatkotestausvaiheessa käytettynä isojen datamassojen kanssa (yli 150000 nimikettä) SentenceTransformer koneoppimismalli *paraphrase-MiniLM-L6-v2* antaa liian suuren määrän tuloksia (yli 800000 nimikeparia), vaikka kynnysarvo olisi asetettu todella korkeaksi (0,999). Lisäksi ison datamäärän käsittely on vaatinut paljon muistikapasiteettia ja saattoi kestää jopa yli tunnin. Muistikapasiteettiongelma saatiin ratkaistua jakamalla dataa eriin (batch) sekä käyttämällä PCA (Principal component analysis) metodia vektoreiden dimensioiden vähentämiseksi. Toista mallia käyttämällä voisi mahdollisesti saavuttaa tarkemmat tulokset duplikaattien haussa. Työn kirjoitushetkellä duplikaattihakutyökalu on otettu toimeksiantajayrityksellä työn alle jatkokehitystä varten ja mahdollista käyttöä pilvialustalla. *paraphrase-MiniLM-L6-v2* mallin toinen sovelluskohde on samankaltaisten (ei identtisten) nimikkeiden haku. Tästä on ollut hyötyä asiakasyrityksen nimikkeiden harmonisointiprojektissa. Jos käsiteltyjen nimikkeiden seassa löytyy samantyyppinen nimike kuin toinen, joiden tietoja ei ole vielä harmonisoitu, kyseisen nimikkeen attribuutit voisi tarkistaa ja täyttää harmonisoidun mallinimikkeen mukaan. Täysiduplikaattien lisäksi työssä löydettiin ratkaisu niin sanottujen piiloduplikaattien hakuun, käyttämällä regex-tekniikka ja Python koodausta. Tämä ratkaisu oli testattu sähkökomponenttinimikedatalla ja yhdistävänä tekijänä oli valittu 7 numeroa sisältävä sähkönumero. Ratkaisun avulla onnistuttiin löytämään useita kymmeniä nimikepareja reilun 2000 nimikkeiden datasettistä, joiden kuvaukset erosivat toisistaan, mutta sähkönumeron perusteella pystyttiin identifioimaan nämä nimikkeet duplikaateiksi. Tämä menetelmä on toiminut hyvin juuri tietynlaiselle datalle. Ruuvinimikkeille, joilla ei ole sähkönumeroa vastaava yhdistävää tekijää, tämä ratkaisu ei antanut käyttökelpoisia tuloksia. Nämä esimerkit kertovat, miten spesifinen on nimikedata, esimerkiksi kahden nimikekuvausten välissä voi olla vain yhden merkin ero ja mallin kannalta nimikkeiden samankaltaisuus olisi yli 99 %, mutta kyseessä ovat ihan eri tuotteet. Sen sijaan ylimääräinen välilyönti, kirjoitusvirhe tai sananlyhenteen käyttö nimikeattribuuteissa voivat laskea duplikaattinimikkeiden välisen samankaltaisuusarvon niin paljon, että malli ei tulkitse niitä saman nimikkeen kaksoiskappaleiksi. Lisäksi nimikeattribuutittien vaihtelevuus tuoteryhmästä riippuen nostaa tehtävän vaikeusastetta. Siksi duplikaattihakuratkaisua, joka olisi hyvin toiminut erityyppisille nimikkeille, voi olla haastavaa löytää. Toimiva ratkaisu voisi olla koneoppimismallin (kuvaustekstin sisällön analysointia varten) sekä regex-tyyppisen metodin (nimikespesifikaatioiden vertailuun) yhdistelmä.

Toinen työn käyttötapaus liittyy laatuvirheiden löytämiseen nimikedatasta. Excel-tiedoston muodossa olevan nimikedatan tarkistukseen oli luotu työkalu. Työkalun tarkistustoiminnot

pohjautuvat Python-skriptiin, joka lataa nimiketiedot Excelistä pandas dataframeen, suorittaa tarkistustoiminnot erillisten funktioiden avulla ja palauttaa tarkistussarakkeet alkuperäiseen Excel-tiedostoon. Skriptin ajo käynnistetään erillisessä .xlsm-tiedostossa olevalla VBA-makrolla. Samassa tiedostossa on omat makrot ja napit virhesolujen värjäystä varten. Virhesolujen värikoodaukset huomattavasti helpottavat tarkastustyötä. Työkalun tarkistustoiminnot ovat pitkälti tarkistettavan datan vertailua yritysstandardivaatimuksia vastaan. Todettiin, että tarkistustyökalu tehostaa ja helpottaa nimikkeiden harmonisointia, siksi se oli otettu käyttöön toimeksiantajayrityksessä jo kehitysvaiheessa. Arvioitu käytännön hyöty työkalun käytöstä on noin 5–10 % säästö rikastustiimin työajasta. Työkalua on tarkoitus käyttää muidenkin asiakasyrityksen yksiköiden nimikedatan käsittelyssä.

Opinnäytetyssä käsitelty nimikehallintaan liittyvät haasteet ovat tavalla tai toisella yleisiä lähes jokaiselle yritykselle koosta ja toimialasta riippumatta. Yhä useammassa organisaatiossa todetaan hallitsemattomasti kasvavan nimikedatan ongelma vakavaksi, isoja investointeja ja korjaustoimenpiteitä vaativaksi. Nimikedatahallinnan ongelmiin ei ole olemassa ratkaisua, joka voisi sopia kaikentyyppiselle datalle sekä kaikille yrityksille. Kuitenkin analysoimalla yrityksen dataa ja kommunikoimalla nimikkeitä käsittelevien sidosryhmien kanssa on mahdollista etsiä ja löytää keinot varmistamaan ja parantamaan nimikedatan laatua. Yritykset, joissa panostetaan datan laatuun, huolletaan ja ylläpidetään nimikedataa säännöllisesti, tulevaisuudessa saavat etumatkaa kilpailijoihin, jotka joutuvat jatkuvasti kampailemaan huononlaatuisen nimikedatan kanssa.

Lähteet

Fitzsimons, M. 2022. Onko yrityksesi master data kunnossa? Näin pääset alkuun ydintietojen hallinnan kehittämisessä. twoday Biit. Viitattu 24.8.2023. Saatavissa https://www.biit.fi/biit-fiid-winter-22-release/onko-yrityksesi-master-data-kunnossa/

Google for Education 2023. Python Regular Expressions. Viitattu 10.1.2024. Saatavissa

https://developers.google.com/edu/python/regular-expressions

ite wiki. Mitä toiminnanohjausjärjestelmä (ERP) tekee? Viitattu 30.8.2023. Saatavissa https://www.itewiki.fi/opas/mita-toiminnanohjausjarjestelma-erp-tekee/

Jokipii, K. 2023. Mikä on ERP-järjestelmä? Katso ERP:n 3 tärkeintä hyötyä. Innofactor. Viitattu 1.9.2023. Saatavissa https://blog.innofactor.com/fi/mika-on-erp-jarjestelma

Mudadla, S. 2023. What is sentence transformer. Why sentence transformer is useful when you have less resources. Medium. Viitattu 28.12.2023. Saatavissa https://medium.com/@sujathamudadla1213/what-is-sentence-transformer-why-sentence-transformer-is-useful-when-you-have-less-resources-13499f7c09bd

Nanyang Technological University 2018. Regular Expressions (Regex). Viitattu 10.1.2024. Saatavissa https://www3.ntu.edu.sg/home/ehchua/programming/howto/Regexe.html

Palatz, P., Costiander, J. 2019. Product Information Management - PIM. Tuotetiedonhallinta. Canter. Viitattu 30.8.2023. Saatavissa https://www.canter.fi/wp-content/uploads/2019/12/Tuotetiedonhallinta-opas-2019.pdf

Pello, R. 2018. Design science research — a short summary. Medium. Viitattu 31.8.2023. Saatavissa https://medium.com/@pello/design-science-research-a-summary-bb538a40f669

Peltonen, H., Martio, A., Sulonen R. 2002. PDM Tuotetiedon hallinta. Helsinki: Edita Publishing Oy IT Press.

Pinecone 2021. Sentence Transformers: Meanings in Disguise. Viitattu 28.12.2023. Saatavissa https://www.pinecone.io/learn/series/nlp/sentence-embeddings/

Pykes, K. 2023. Fuzzy String Matching in Python Tutorial. Datacamp. Viitattu 16.2.2024. Saatavissa https://www.datacamp.com/tutorial/fuzzy-string-python

Solidworks PDM Help 2020. Items (For SOLIDWORKS PDM Professional only). Dassault Systems. Viitattu 23.8.2023. Saatavissa https://help.solidworks.com/2020/English/EnterprisePDM/FileExplorer/HelpViewerDS.aspx

?version=2020&prod=EnterprisePDM&lang=English&path=FileExplorer%2fc_items_overview.htm

Tanskanen, A., Nivamo, J. 2017. Savonia-artikkeli: Mikä se ERP-järjestelmä oikeastaan on? Diva. Viitattu 1.9.2023. Saatavissa https://diva.savonia.fi/savonia-artikkeli-mika-se-erp-jarjestelma-oikeastaan-on/

The W. Edwards Deming Institute, 2023. PDSA Cycle. Viitattu 20.9.2023. Saatavissa https://deming.org/explore/pdsa/

Tiuttu, J. 2019. Tuotetiedot ja ERP kuuluvat yhteen. Roima Akatemia. Viitattu 6.9.2023. Saatavissa https://www.roimaint.fi/tuotetiedot-ja-erp-kuuluvat-yhteen/

Trebuňa, P., Kliment, M., Fil'O, M., Marcovič, J., Halčinová, J. 2013. PLM systems, their history and application today in business process modeling. Technical University of Košice, Slovakia. Viitattu 1.9.2023. Saatavissa http://elib.bsut.by/bitstream/handle/123456789/4841/%D0%A2rebuna%20P.129%20-%20134.pdf?sequence=1&isAllowed=y

Väre, T. 2019. Master data. Helsinki: Alma Talent.

Varshney, S. 2023. What is Data Quality? Dimensions & Their Measurement. OvalEdge. Viitattu 14.9.2023. Saatavissa https://www.ovaledge.com/blog/data-quality-metrics