VERSIONHALLINNAN KEHITTÄMINEN BIG DATAAN LIITTYVÄSSÄ ETL KEHITYS- JA YLLÄPITOTYÖSSÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

Tuotantotalouden diplomityö

2022

Tomi Hiltunen

Tarkastajat: Professori Timo Kärri

Tutkijatohtori Lasse Metso

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT

School of Engineering Science

Tuotantotalouden koulutusohjelma

Tomi Hiltunen

Versionhallinnan kehittäminen Big Dataan liittyvässä ETL kehitys- ja ylläpitotyössä

Tuotantotalouden diplomityö

82 sivua, 21 kuvaa, 8 taulukkoa ja 1 liite

Tarkastajat: Professori Timo Kärri ja tutkijatohtori Lasse Metso

Avainsanat: ETL, Versionhallinta, Git, SVN, DataOps

ETL kehitys ja ylläpito on tärkeässä roolissa kaikessa datan hyödyntämisessä. ETL kehityksellä tarkoitetaan prosessia, jossa data kerätään ja muokataan hyödynnettävään muotoon. ETL prosessiin käytetään kaikista tietovarastointiprojektien resursseista jopa 80 %. Työn tavoitteena oli löytää ratkaisukeinoja ETL kehittäjien resurssien ohjaamiseen manuaalisista prosesseista kehittämiseen. Työssä pyrittiin löytämään myös versionhallinnan yleinen toimintamalli, joka toimisi useimmissa projekteissa.

Työ toteutettiin usean osatapauksen tapaustutkimuksena, jossa hyödynnettiin puolistruktu-roitua haastattelumallia. Haastatteluiden perusteella luotiin nykytilan katsaus. Nykytilan perusteella valikoitiin kehitettävät kohteet suurimpiin ajansäästöllisiin kohteisiin. Teoriaosuudessa tutkimustietoa ETL kehityksen versionhallinnasta ei ollut, joten työssä jouduttiin hyödyntämään ohjelmistokehityksen tutkimuksia. Näistä tutkimuksista ja teoriasta saatuun tietoon hyödynnettiin ETL kehityksen ominaispiirteitä.

Työn lopputuloksena syntyi teoriatiedon perusteella luotu versionhallinnan malli ja toimintatapa. Yleisessä versionhallinnan mallissa otettiin huomioon ETL työkalujen luomat haasteet, kuten rakenteellinen tekstitiedosto. Tähän lisäyksenä empiriaosuuden pohjalta kehitettiin konseptitodistus työkalusta. Konseptitodistus on todistus ohjelman toteuttamiskelpoisuudesta. Työkalulla saatiin luotua helpommin käytettävä, aikaa säästävä ja virheitä vähentävä toimintakokonaisuus.

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

School of Engineering Science

Industrial Engineering and Management

Tomi Hiltunen

Developing version control in Big Data related ETL work

Master's thesis

2022

82 pages, 21 figures, 8 tables and 1 appendix

Examiners: Professor Timo Kärri and Postdoctoral Researcher Lasse Metso

Keywords: ETL, Version control, Git, SVN, DataOps

ETL development and maintenance play a significant role in all data utilization. ETL development refers to the process of collecting and modifying data into the format to be utilized. Up to 80 % of all resources in data warehouse projects are used for the ETL development. The aim of the work was to find solutions for directing ETL developers' resources from manual processes to actual development. The work also sought to find a general operating model for version control that would work on most projects.

The work was conducted as a case study in several sub-cases, using a semi-structured interview model. Based on the interviews, an overview of the current situation was created. Based on the current situation, targets to be developed were selected for the largest time-saving phases. In the theory section, there was no research data on the version control of ETL development, so this thesis had to utilize software development studies. The characteristics of ETL development were added to the information obtained from these studies and theories.

The result of the thesis was a version control model and operating method created based on theoretical knowledge. The general version control model considered the challenges created by ETL tools, such as a structured text file. As an addition to this, a proof of concept of the tool was developed based on the empirical section. Proof of concept is a proof of the feasibility of the program. The tool was used to create a more accessible, timesaving and error-reducing workflow.

Sisällysluettelo

Liitteet

Liite 1. Haastattelurunko

Kuvaluettelo

Taulukkoluettelo

1 Johdanto

Digitalisaation ja datan kasvavan hyödyntämisen takia ETL kehityksen merkitys tulee kasvamaan yritysten toiminnassa. Datan hyödyntämiseen tarvitaan prosessia, jolla muutetaan lähteistä saatava data luotettavaan ja yhdenmukaiseen käytettävään muotoon. Tätä prosessia kutsutaan ETL prosessiksi ja se on datan hyödyntämisen tärkein osa. Tällä hetkellä ETL kehitykseen käytetään tietovarastointiprojekteissa noin 70–80 % resursseista (Novak & Rabuzinin, 2014). Tästä syystä kehityksen ja lopputulosten kannalta merkityksettömien manuaalisten työosuuksien poistamisen pitäisi olla suuressa huomiossa, kun mietitään miten ETL kehityksen toimintatapaa ja ympäristöä kehitetään. Täten työn aihepiiriksi valittiin versionhallinnan käsittely ETL kehityksen resurssisäästöissä.

ETL kehityksessä ominaista on tiedostotyyppien luomat haasteet versionhallinnassa. Koska tiedostotyypit ovat rakenteellisia tekstitiedostoja, voitaisiin teoriassa tiedostot yhdistää. Eri ETL työkaluilla tuotettujen tiedostojen tiedostotyypit voivat kuitenkin olla ohjelmakohtaisia, joten tässä työssä ei pyritä kehittämään tällaista työkalua tai toimintatapaa.

Kerätessä aineistoa teoriaosuutta varten huomattiin, ettei ETL kehityksen versionhallintaa olla tutkittu muuten kuin teoreettisella tasolla. Näin ollen työssä on jouduttu käyttämään ohjelmistotuotannon vastaavaa, joskin tähän on lisätty ETL työkalujen ja kehitystavan ominaisuuksia sovellettavan teorian keräämiseksi. Aihepiiriin liittyy ohjelmistokehityksessä tunnettu termi DevOps, jonka pääasiallisena päämääränä oli nopeuttaa ohjelmistokehityksen prosessia ja tuoda kehitystiimiä yhtenäisemmiksi (Ebert, Gallardo, Hernantes ja Serrano. 2016) Datan hyödyntämisessä käytetään termiä DataOps, jonka periaate on samanlainen kuin ohjelmistokehityksen termillä. Suuri kansainvälinen ICT alan tutkimus- ja konsultointiyritys Gartner (2018) on listannut DataOpsin yhdeksi innovaatioiden laukaisijaksi. Gartner toisaalta on myös lisännyt, että DataOps on uusi käytäntö, jolla ei ole standardeja tai runkoa. Tältä osin voidaan siis sanoa aiheen olevan trendikäs, joskin vähän tutkittu tai kehitetty toimintatapa. DataOpsin pääperiaate on jatkuvan kehityksen ja integraation malli, mutta koska ETL työkalujen tiedostotyypit aiheuttavat yhdistämisongelmia ei tätä voida vielä pitää validina työn nopeuttamiseksi useamman kehittäjän ympäristöissä.

Harvinder Atwal (2019 s.162) mainitsee kirjassaan ***Practical DataOps: Delivering Agile Data Science at Scale,*** että useissa organisaatioissa on haasteellista julkaista tuotantoon useita versioita viikossa, koska usein tuotantoon vienneissä kohdataan ongelmia. Ongelmia syntyy, koska dataputket ja taulut ovat toisistaan riippuvaisia ja pienetkin virheelliset muutokset saattavat aiheuttaa suuria muutoksia muissa tauluissa. Tämä johtaa tulipalojen sammuttamiseen, taulujen palautuksiin sekä koko järjestelmän käyttökatkoihin. (Atwal, 2019) Tästä syystä työssä pyritään keskittymään vain versionhallinnan aspektiin, jonka käyttöönotto voi olla käyttäjille haastavaa ja toimintatapojen noudattaminen tukeutua täysin ennalta määriteltyyn ohjeistukseen.

1.1 Tavoitteet ja rajaus

Tämän tutkimuksen päätavoitteena oli löytää kohdeyrityksen asiakkaille yhtenäinen tapa toimia versionhallinnan kanssa ETL kehityksessä, sekä kehittää työkalu, jonka avulla versionhallinnan manuaalista työtä ja inhimillisiä käyttäjävirheitä saadaan vähennettyä. Näin ollen ETL kehittäjien aikaa saadaan säästettyä epäoleellisesta työstä ja suunnattua kohti kehittämistä. Epäsuorana tavoitteena oli myös yhtenäistää versionhallinnan rakennetta ja toimintaa. Täten häiriötilanteita kohdatessa toiminta ja korjaustekniikat ovat samat ja versionhallinta toimii ETL kehitystiimin tukena ja keskeisenä työkaluna kuten sen on tarkoitettu.

Tutkimuksessa tarkastellaan kohdeyrityksen asiakkaiden nykytilannetta haastattelemalla tietovarastoasiantuntijoita. Haastatteluista syntyvää tilannekuvaa analysoimalla pyritään löytämään kehitettäviä osa-alueita ja ajankäytöllisiä säästökohteita. Aihe on mielenkiintoinen, sillä tiedon ja data-analytiikan käyttö yleistyy yhä enemmissä määrin koko maailmassa ja ETL kehitys on tässä prosessissa keskeisessä roolissa datan keräämisen kanssa. Kuten sanottu, Novakin ja Rabuzinin (2014) mukaan ETL kehitykseen käytetään tietovarastointiprojekteissa noin 70–80 % resursseista, joten ylimääräisten resurssien säästämiseksi tämänkaltainen projekti on sopiva ja ajankohtainen.

Tässä tutkimuksessa kartoitetaan asiakasyritysten nykytilaa, jonka pohjalta kehitetään yhtenäinen toimintatapa ja työkalu, jolla saadaan tuotettua lisäarvoa tietovarastoprojekteissa, sekä asiakkaalle että konsultointiyritykselle. Tämä tutkimus jakaantuu teoriaosuuteen, haastatteluihin perustuvaan tapaustutkimukseen sekä lopputulosten tarkasteluun.

Teoriaosuudessa esitetään työssä hyödynnettävät käsitteet ja organisaation ETL ympäristön ihannetilanne. Tämä osuus on toteutettu tieteellisiin julkaisuihin ja teoksiin perustuen. Tapaustutkimuksessa on käsitelty useampia kriteereiden täyttämiä asiakasyrityksiä, jonka tilannekuvan pohjalta lopputuloksien toinen osa tehdään. Lopputuloksissa käsitellään teoriaosuuden pohjalta kehitettyä toimintatapaa sekä konseptitodistusta työkalusta, jolla kehittäjien työaikaa saadaan suunnattua kehitykseen. Konseptitodistus on tässä tapauksessa versionhallinnan ja ETL kehityksen tueksi kehitetyn ohjelman toteuttamiskelpoisuuden todistus.

Työhön on valikoitu päätutkimuskysymyksiksi

1. Minkälainen versionhallintamalli sopii parhaiten ETL kehitysympäristöihin, joissa on useampi kehittäjä?
2. Kuinka saadaan kehittäjien aikaa siirrettyä manuaalisista vaiheista itse kehittämiseen?

Näistä ensimmäisen kysymyksen tavoitteena on käsitellä yleisesti ETL kehityksen ominaispiirteitä, jotka estävät normaalin ohjelmistotuotannosta tutun versionhallinnan toimintatavan. Näitä piirteitä ymmärtämällä pyritään kehittämään yksinkertaisella tasolla malli, jolla saadaan yhtenäinen toimintatapa versionhallinnan käyttöön. Toisella pääkysymyksellä tavoitellaan kehittäjien ajankäytöllistä säästämistä versionhallinnan ja kehitysympäristöjen välillä tapahtuvien siirtojen parissa. Näihin päätutkimuskysymyksiin pyritään löytämään vastauksia, sekä ymmärtämään tutkittavaa aihepiiriä paremmin pohjautuen työn teoriaosuuteen sekä haastatteluissa ymmärtämällä yritysten nykytilannetta seuraavilla apukysymyksillä.

1. Miten versionhallintaa hyödynnetään asiakasyritysten ETL kehityksessä?
2. Mitkä ovat yritysten versionhallinnan työläimmät ja virhealttiimmat osiot ja mitä haasteita versionhallinta tuo kehitysprosessiin?
3. Minkälainen on kehityksen kokonaisprosessi, määrittelystä ylläpitoon?

Ensimmäisellä apukysymyksellä pyritään löytämään lähtötaso versionhallinnan osalta, eli määritellään nykyiset toimintatavat ja -mallit sekä ympäristö, jossa versionhallinta tapahtuu. Ensimmäinen apukysymys myös määrittelee millä tavalla kohdeorganisaatio hyödyntää versionhallintaa kehityksen tukena. Toisessa apukysymyksessä pureudutaan tarkemmin kehitysympäristön luomiin haasteisiin ja versionhallintaan kuluvaan ajankäytölliseen

jakaumaan. Tavoitteena tällä apukysymyksellä on määritellä kehitystä vaativat kohdat tarkemmalla tasolla sekä virheiden että ajankäytön suhteen. Viimeinen apukysymys tuo yleistä tietoa kehitys- ja ylläpitoprosessista, kehitysympäristöjen välisten siirtojen haasteellisuudesta tai ominaispiirteistä sekä luo yleiskuvaa ETL kehityksen haasteista.

Työtä on rajattu siten, että haastatelluissa käsiteltävien yrityksien tulee hyödyntää dataa laajemmin. Tällä tarkoitetaan sitä, että ETL kokonaisuuksia on suuri määrä, kehitysympäristöjä on useampia ja ETL kehitystä tehdään jollain markkinoilla olevalla ETL-työkalulla. Näin ollen yritykset ovat suurempia, käyttävät kohdeorganisaation tietovarastoinnin asiantuntijoita sekä käyttävät versionhallintaa tai varmuuskopioita jollain tasolla. ETL-työkalun vaatimus on määritelty siksi, että käsin koodatessa ETL kokonaisuus luodaan jollain ohjelmointikielellä. Täten kokonaisuus on helpommin muokattavissa, yhdisteltävissä ja siirreltävissä kehitysympäristöistä toisiin. Työkalu luo siis omat haasteensa kehitykseen ja ominaispiirteensä versionhallinnan käyttöön.

1.2 Tutkimuksen menetelmät ja aineisto

Tämä tutkimus on toteutettu laadullisena eli kvalitatiivisena tutkimuksena. Laadullisessa tutkimuksessa aineisto pyritään keräämään monia näkökulmia antavista lähteistä, joiden avulla saadaan luotua kattava kuvaus tutkittavasta ilmiöstä. (Aira, 2005) Aira lisää, että laadullinen tutkimus sopii hyvin asenteiden ja uskomusten tutkimiseen, jossa tutkittava aihepiiri on vähemmän tutkittu ja siitä ei vielä tiedetä paljoa. Pitkärannan (2010) mukaan laadullisen tutkimuksen tavoitteena on vastata kysymyksiin miksi ja miten. Täten voidaan suunnitella uusia tuotteita, menetelmiä tai toimintatapoja organisaatiolle, joten se soveltuu tällaisen aiheen selvittämiseen erinomaisesti.

Pitkärannan (2010) mukaan laadullisen tutkimuksen aineiston keruun yleisin tapa on haastattelu, jonka rakenne voi olla strukturoitu, puolistrukturoitu, teemahaastattelu tai avoin. Strukturoidussa haastattelussa kysymysten muotoilu on kaikille haastateltaville sama ja haastatteleva antaa vastausvaihtoehdot. Puolistrukturoidussa haastattelussa kysymykset ja muotoilu on sama, mutta vastaukset ovat avoimia eli haastateltava vastaa kysymyksiin omin sanoin. Teemahaastattelussa haastattelija määrittelee haastattelun aihepiirin, mutta

kysymykset ja haastattelun rakenne rakentuu haastattelun edetessä. Tässä tutkimuksessa käytetään haastatteluihin puolistrukturoitua haastattelumallia.

Tämän tutkimuksen empiriaosuus on toteutettu useamman otoksen tapaustutkimuksena, joka toteutetaan haastattelemalla Enfo Oyj:n asiakasyrityksiä. Haastatteluiden tavoitteena on selvittää ETL-kehityksen versionhallinnan tunnuspiirteitä sekä koota tieto yhdeksi kokonaisuudeksi, jolla on yleisemmällä tasolla hyödynnettävää merkitystä. Tapaustutkimusta, jossa on usea osatapaus, kutsutaan myös monitapaustutkimukseksi, jonka tarkempi esittely ja määrittely esitetään työn viidennessä luvussa.

Kuten aiemmassa kappaleessa mainittiin, aineiston keräämiseen on käytetty kohdeyrityksen asiakasyrityksiä, joissa asiantuntijuutta on käytetty tietovarastoinnissa ja ETL kehityksessä. Haastatteluissa mukana on ollut kohdeorganisaation kokeneita tietovarastoinnin asiantuntijoita, joten käsitteet ja määrittelyt ovat sekä haastattelijalla että haastateltavilla samat.

1.3 Raportin rakenne

Tutkimus rakentuu johdannon jälkeen teoriaosasta, tutkimusmenetelmien ja -aineiston esittelystä, tutkimustulosten esittelystä, varsinaisista lopputuloksista ja yhteenvedosta. Työn teoreettisen taustan muodostavat luvut 2–4. Näissä kappaleissa käsitellään teoreettisella tasolla työn aihepiiriin liittyvät tärkeät käsitteet ja näiden suhde toisiinsa.

Luvussa 2 esitellään Big datan määritelmä, sen tarkoitus ja miten se liittyy ETL prosessiin. Kolmannessa luvussa käydään laajemmalla tasolla ETL prosessin määritelmä, sen vaiheet ja Ralph Kimballin määrittelemän hyvän ETL ympäristön ja työkalun piirteet. ETL kehittämisen luvussa käydään läpi myös ETL kehityksen kannalta merkityksellisten vaiheiden yksityiskohtia kuten testaaminen ja työkalut. Neljännessä luvussa käsitellään versionhallintaa. Teoreettisella tasolla käsitellään versionhallinnan kehittymistä nykyiseen muotoonsa sekä versiohaarojen merkitystä ETL kehityksessä.

Teoriaosan jälkeen luvussa viisi esitellään tutkimuksen toteutustapa, tutkimusprosessi sekä esitellään tutkimuksessa käytetty tutkimusmenetelmä ja aineistonkeruutapa. Esittelyjen jälkeen luvussa 6 raportoidaan empiriaosuudessa tehdyn haastattelun tulokset ja näiden tuloksien pohjalta määritellyt kehityskohteet. Kehityskohteiden määrittelyn jälkeen luvussa 7.

esitellään työn aikana muodostuneet lopputulokset ja näitä lopputuloksia verrataan johdannossa määriteltyihin tavoitteisiin. Viimeisessä luvussa esitetään koko työn johtopäätökset.

2 Big Data

Yleisesti suomen kielessä Big datasta käytetään englannin kielistä termiä, mutta sille on myös tietotekniikan termitalkoissa kehitetty suomenkielisetkin vähemmän tunnetut termit massadata ja iso data. Tietotekniikan termitalkoiden (2013) määrittelyssä käsite on kuvattu seuraavasti *”data, jota on paljon, jota tulee nopeasti lisää ja joka on muodoltaan vaihtelevaa”.* Tietotekniikan termitalkoissa myös mainitaan, että Big dataa pyritään keräämään, säilyttämään ja analysoimaan merkityksellisen informaation löytämiseksi. Tämä aiheuttaa haasteita, sillä tietomäärät ovat valtavia, ne ovat rakenteelliselta muodoltaan erilaisia ja käsittelyyn vaaditaan laaja ammattitaito. Manyika et al. (2011) listaa Big datan käyttöön vaadittaviksi tekniikoiksi yhteensä 26 eri tekniikkaa, kuten seuraavat: A/B testaus, assosiaatiosääntöjen oppiminen, luokittelu, klusterianalyysi, datan yhdistäminen ja integrointi, koneoppiminen, neuroverkot, hahmontunnistus ja ennustava mallinnus.

Big dataa käsitellään tässä työssä siksi, että se liittyy vahvasti ETL kehitykseen. ETL on prosessi saada Big datasta hyödyllistä ja käsiteltyä tietoa päätöksenteon tueksi tai koneoppimismallien käyttöön. Voidaan olettaa, että kun ETL prosesseja käytetään suuremmassa mittakaavassa niin lähteenä käytetään yleisesti Big dataa.

2.1 Termin kehitys

Big datan määritelmä on ollut pitkään suhteellisen epäselvä ja se on muuttanut muotoaan vuosien varrella. Vuonna 1997 termiä käytettiin suurten tietokokonaisuuksien visualisoinnissa. Seuraavina vuosina vuoteen 2001 asti määritelmä on vaihdellut laitteistoon liittyvästä termistä data louhinnan piiriin ja sieltä edelleen tilastotieteeseen. (Ylijoki & Porras, 2016)

Vuonna 2001 Doug Laney asetti Big datalle kolme määrittelevää dimensiota. Tätä on kutsuttu 3V:n määritelmäksi sillä se koostuu sanoista Volume, Velocity ja Variety. Big dataa on siis data, joka koostuu suuresta datamassasta, se kasvaa nopeasti ja sisältää rakenteellisesti erilaisia tiedostoja kuten kuvat, videot, tekstit tai äänitteet. (Gandomi & Haider, 2014)

Vuoden 2001 alkuperäisen laajasti levinneen määritelmän lisäksi useat alan toimijat ovat lisänneet 3V:n määritelmään omia lisäyksiään. Tällaisia lisäyksiä on tehnyt esimerkiksi IBM, joka lisäsi määritelmään myös dimensiot Variability ja Value. Variability:llä tarkoitetaan, sitä että täytyy tietää datan luomisen konteksti, jotta sitä voidaan pitää luotettavana. (Jain, 2016) IBM käyttää esimerkkinään tilannetta, jossa sama kirjainlyhenne tarkoittaa eri ihmisen ja ammattiryhmän kohdalla eri asioita. Value on dimensio, joka määrittelee Big dataa siten, että datalla täytyy olla arvoa. SAS puolestaan on lisännyt omaan Big datan määritelmäänsä sanat Variability ja Veracity.

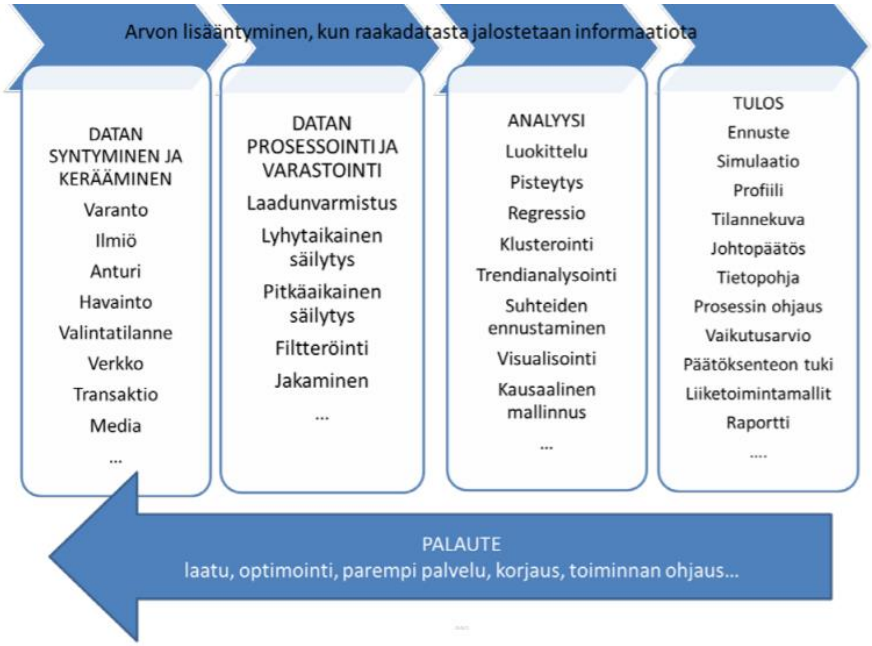
Koska Laneyn kuvauksesta puuttuu liiketoiminnallinen puoli, voidaan nykypäivänä hyvänä Big datan kuvauksena pitää viiden V:n ja yhden C:n määrittelyä. (Oguntimilehin & Ademola, 2014)

- Volume (Määrä)
- Velocity (Nopeus)
- Variety (Rakenteelliset erot)
- Variability (Muuttuvuus)
- Value (Arvo)
- Complexity (Monimutkaisuus)

2.2 Big datan käyttö

Tilastokeskuksen vuonna 2018 tekemän kyselytutkimuksen mukaan 19 % yrityksistä on käyttänyt hyväksi Big dataa. Eniten Big dataa hyödyntänyt toimiala kyselytutkimuksessa on ollut informaation ja viestinnän toimiala (42 %) ja vähiten vähittäiskaupan toimialalla (10 %)

Liikenne- ja viestintäministeriön (2014) tekemän selvityksen mukaan Big datan hyödyntämisellä tavoitellaan toiminnan optimointia, optimoinnista saatavia säästöjä ja sitä käytetään ennustamiseen, korrelaatioiden havaitsemiseen. Näistä kaikki johtavat paremman päätöksenteon tukeen. Samalla selvityksessä mainitaan, kuinka *”Tiedon yhä paremmasta hyödyntämisestä on tulossa ehdoton menestymisen edellytys”*



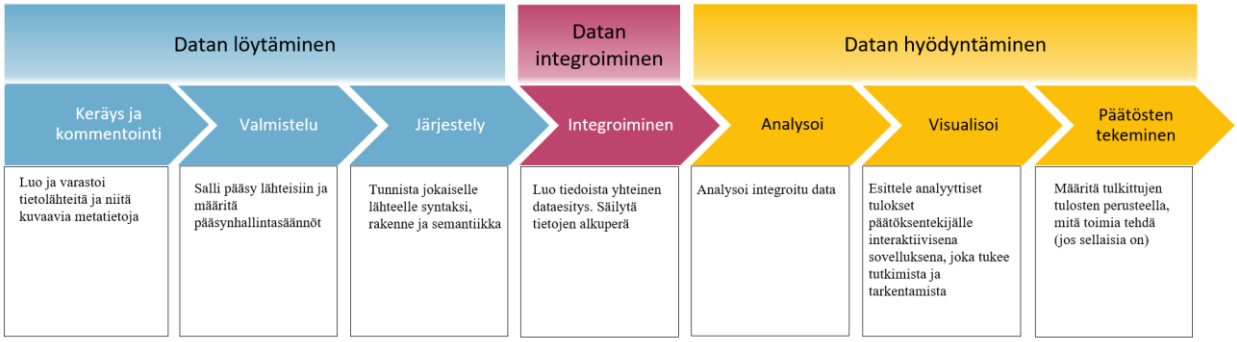
Kuva 1. Datan hyödyntämisen arvoketju (Liikenne- ja viestintäministeriö, 2014)

Datan hyödyntämisen arvoketjussa (Kuva 1) nähdään asteittain Big dataan liittyvät toimenpiteet kuten luokittelu ja visualisointi, jotka johtavat ennusteeseen tai johtopäätöksiin. Kuvassa nähdään myös omana osanaan datan prosessointi ja varastointi, jota voidaan pitää ETL-prosessina. Tässä vaiheessa syntynyt data poimitaan, muutetaan ja ladataan joko lyhytaikaiseen tai pitkäaikaiseen säilytykseen tietokantaan. Tässä arvoketjussa ETL prosessi on tärkeässä roolissa, sillä huono lähtödata tai väärin tehdyt muutokset johtavat vääriin johtopäätöksiin. (Lindholm, 2021)

Sravanthi ja Tatireddy (2015) listaavat tekstissään toimialoja tai toimintoja, joilla pystytään käyttämään tai käytetään Big dataa hyödyksi. Näitä hyödyntäjiä on listattu 17, joista esimerkiksi sijoitusmaailmassa Big dataa ja tietokantojen analysointia käytetään väärinkäytön havaitsemiseen ja kuluttajahyödykkeissä, joissa tuottaja kerää jatkuvasti tietoa kuluttajien mieltymyksistä ja mielipiteistä. Näin saadaan parempi tilannekuva siitä, mitä kuluttajat haluavat ja miksi jokin tuote myy paremmin kuin toinen.

3 ETL-kehittämisen tarkastelu

ETL terminä tulee sanoista Extract – Transform – Load, eli suomennettuna poiminta, muunto ja lataus. Nämä 3 osiota eritellään ja esitellään tarkemmin seuraavissa kappaleissa. Varsinainen ETL prosessi voidaan sijoittaa H. Gilbert Millerin ja Peter Morkin (2013) esittelemän kuvan kohtiin datan keräys ja kommentointi, valmistelu, järjestely ja lopulta integroiminen. Nämä 4 vaihetta tehdään viimeistä kokonaisuutta eli datan hyödyntämistä varten.



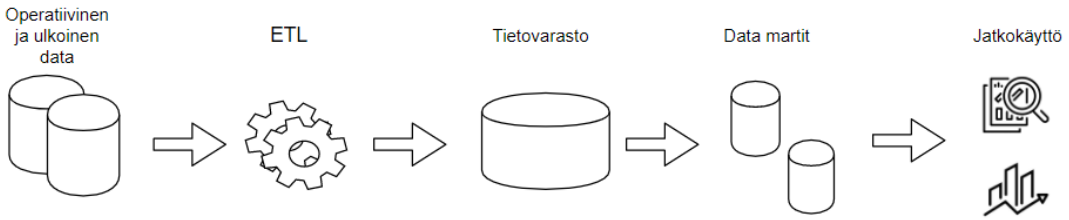
Kuva 2 Datan arvoketju (Miller & Mork, 2013)

Kuten kuvasta 2 nähdään, ETL prosessi pitää sisällään tietolähteiden keräämisen ja näihin liittyvien metadatojen hallinnan. Tietolähteiden keräämisen jälkeen prosessissa valmistellaan tietolähteet käyttöoikeuksien hallinnalla ja järjestelyllä, jossa lähteiden rakenne, semantiikka ja syntaksi määritellään. Semantiikalla tarkoitetaan eri kenttien ja koko taulun merkitystä. Taulujen merkityksellä on suuri rooli, koska eri lyhenteet voivat tarkoittaa eri konteksteissa eri asioita. Datan integroimisessa data käsitellään ja siihen luodaan surrogaattiavaimia sekä yhdistetään dimensioavaimia. Surrogaattiavaimilla tunnistetaan jokainen rivi omakseen ja dimensioavaimilla kyseinen taulu pystytään liittämään dimensiodataan, joka voi pitää sisällään esimerkiksi asiakkaan tiedot.

3.1 ETL:n Määritelmä

ETL prosessi on tiedon integroinnin ydinosa, joka liittyy tyypillisesti tietojen varastointiin. ETL-prosessin aikana tiedot poimitaan OLTP (**On**line **T**ransaction **P**rocessing) -tietokannoista, muunnetaan vastaamaan tietovarastomallia ja ladataan tietovarastoon. (El-Sappagh

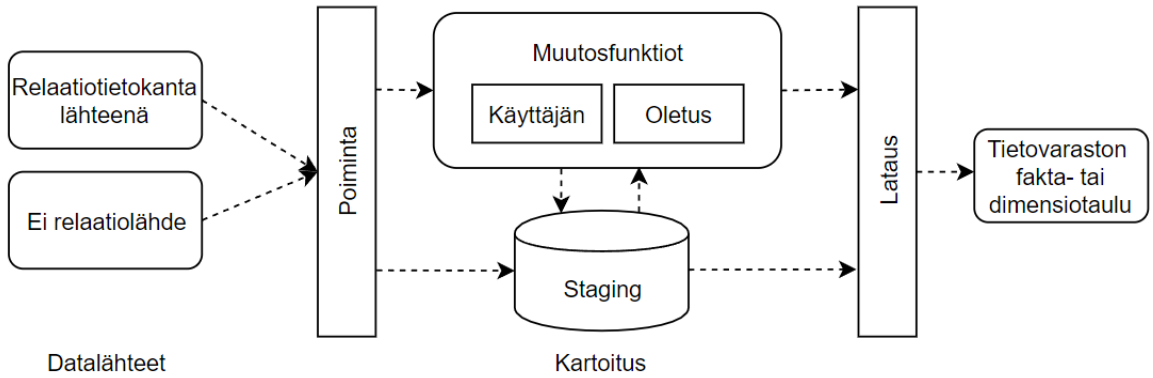
et al. 2011) Kuvassa 3 nähdään datan hyödyntämisen prosessi, jossa ETL sijoittuu datan keräämisen ja tietovarastoinnin väliin.



Kuva 3 ETL prosessin sijainti datan käytössä (Yhdistetty Kimball & Ross, 2013; Golfarelli & Rizzi, 2009)

ETL-työkalut siis poimivat datan lähteistä, muuntaa ja puhdistaa sen liiketoimintasääntöjen mukailemiin muotoihin ja lataa sen viimeisenä haluttuun tietovarastoon. Gour et al. (2010) mukaan ilman ETL prosessia tietovarastoissa ei olisi lainkaan strategista tietoa. Vaikka datan poimiminen on hallitsevassa roolissa ETL-työkalujen käytössä, sitä käytetään myös tiedonsiirtoprosessissa ja siirtämisessä analyysitoiminnoista operatiivisiin systeemeihin.

Kuva 4 näyttää prosessikuvauksen, mitä kuvan 3 ETL prosessi pitää sisällään. Kuvassa nähdään, että prosessi alkaa tiedon lähteestä, siirtyen poimintaan ja siitä kartoitukseen. Kartoitus tarkoittaa kahden vastaavuuden luomista eri datamallin kenttien välille (Haavisto, 2015). Viimeisenä vaiheena on lataus, jossa poimittu ja käsitelty data ladataan tietovarastoon fakta- tai dimensiotauluna.



Kuva 4 ETL prosessikuvaus (El-Sappagh et al. 2011)

Gour et al. (2010) mukaan tehokkaiden ja luotettavien ETL prosessien käyttöönotossa on useita ongelmakohtia. He ovat listanneet ongelmakohdiksi

- Tekniset ongelmat datan liikuttamisessa, integroimisessa ja muuttamisessa eri ympäristöjen välillä
- Lyhyet latausikkunat ja pitkät latausajat
- Ne ovat epäjohdonmukaisia
- Liiketoimintasääntöjen vaikea ylläpidettävyys
- Liiketoimintasääntöjen puutteellinen esittely loppukäyttäjille
- Lähdejärjestelmien puutteet kriittisissä tiedoissa
- Heikot kyselyjen suoritukset

Sekä lisänneet, että mikäli useasta lähteestä poimittua lähtödataa ei ole puhdistettu, poimittu kunnolla, muutettu oikeaan muotoon tai integroitu oikealla tavalla ei varsinainen kyselyprosessikaan olisi mahdollinen.

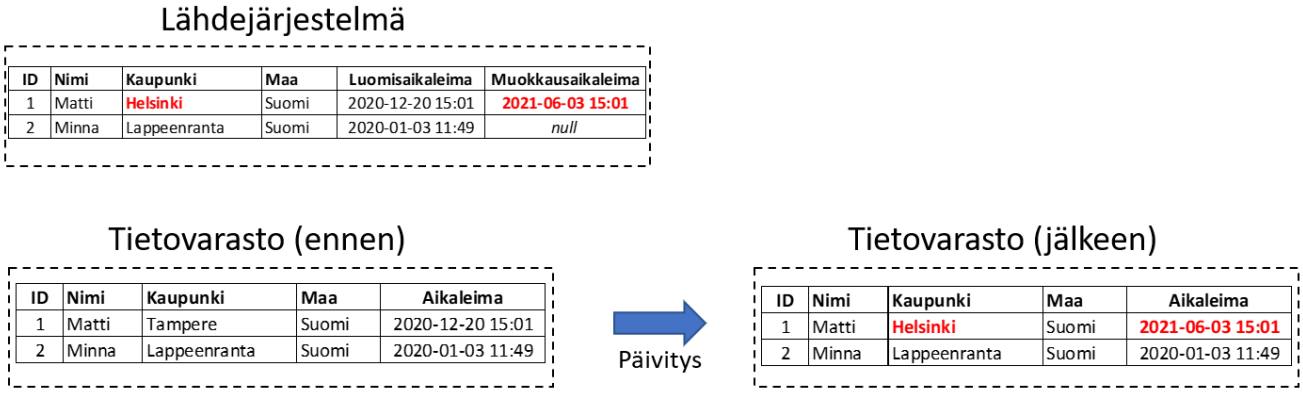
3.1.1 Poiminta / Extract

Kakishin ja Kraftin (2012) mukaan lähdejärjestelmän tiedot voivat olla monimutkaisia, joten merkityksellisten tietojen määrittäminen on hyvin vaikeaa. Louhintaprosessien suunnittelu ja luominen on erittäin aikaa vievää. Jotta tiedot olisivat ajan tasalla tietovarastossa, tiedot on poimittava säännöllisesti ja useasti. He ovat esitelleet poimimiseen kaksi loogista menetelmää, joihin poimintaprosessi voidaan jakaa. Nämä ovat sekä Kakishin ja Kraftin (2012), että Kimball & Rossin (2013) mukaan kokonainen poiminen eli alkulataus ja inkrementaalinen poimiminen.

Alkulataus on ensimmäinen kerta, kun tiedot kootaan eri operatiivisista lähteistä ja ladataan tietovarastoon. Tämä prosessi tehdään vain kerran tietovaraston rakentamisen jälkeen, jolloin se täytetään valtavalla määrällä lähdejärjestelmien tietoja. Alkulataus tehdään ensimmäisellä kerralla, mikäli poimittavaa dataa on jo olemassa suuria määriä. Tässä

poimintatyypissä ei tarvitse seurata lähdejärjestelmään tehtyjä datamuutoksia sillä niitä ei tulla enää poimimaan uudelleen ja muokkaamaan kohdejärjestelmään. (Kakish & Kraft, 2012)

Inkrementaalista poimintaa kutsutaan CDC:ksi (**C**hanged **D**ata **C**apture), joka on edellisen tavan vastakohta, eli vain muuttunutta dataa poimitaan. Inkrementaalisessa poiminnassa ETL-prosessit päivittävät tietovarastoa jaksollisesti lähdejärjestelmiin edellisen poiminnan jälkeen muokatuilla ja lisätyillä riveillä. (Farmer, 2017) CDC tallentaa vain muuttuneet tiedot edellisen poiminnan jälkeen käyttämällä useampia erottavia tekniikoita, kuten Audit-kenttiä, tietokantalokia ja järjestelmän päivämäärää. (El-Sappagh, 2011) Kakishin ja Kraftin (2012) mukaan useimmat ETL-työkalut eivät käytä tätä CDC-mekanismia, vaan ne poimivat kaiken datan lähdejärjestelmistä ja vertaavat tätä dataa aiemmasta poiminnasta poimittuihin tietoihin tai taulukoihin muutosten tunnistamiseksi. Kuvassa 5 esitetään yksinkertaistettuna, kuinka inkrementaalisella latauksella päivitetään olemassa olevaa dataa poimimalla pelkästään muuttunut rivi 1 ja päivittämällä tietovarasto tuon poimimisen pohjalta.



Kuva 5. Inkrementaalisen latauksen kuvaus

Jarrett Goldfedder (2020) mainitsee kirjassaan ***Building a Data Integration Team***, että poimintavaiheessa ETL työkalulla tai käsin koodatessa kehittäjällä on pääasiassa 3 tehtävää (Kuva 6) jotka ovat:

1. Yhdistimien asettaminen
2. Väliaikaisten tietojen tallentaminen
3. Saapuvan datan vahvistus

Poiminta	Muutos	Lataus
• Yhdistimien asettaminen • Väliaikaisten tietojen tallentaminen • Saapuvan datan vahvistus	• Datan muokkaus • Datan eheyden analysointi • Loppudatan vahvistus	• Yhdistimien asettaminen • Lataustyyppien päätteleminen • Virhetilanteissa alkutilanteen palauttaminen

Kuva 6 ETL työkalun pääasialliset tehtävät (Goldfedder, 2020)

Poimimisvaiheessa tiedot haetaan lähdejärjestelmistä kuten pilvestä, asiakirjasta, tietokannasta tai vaikkapa toiminnanohjausjärjestelmästä. Näihin kaikkiin tarvitaan jonkinlainen yhdistin tai liitin, joka yleensä on tiettyyn järjestelmään kehitetty ja soveltuva tulkki/kääntäjä. Yhdistimen avulla järjestelmä voi lukea tietoja, jotka ovat toisessa, ei yhteensopivassa järjestelmässä. (Marinos et al. 2010) Goldfedder kuitenkin lisää, että nykyään ETL työkalujen toimittajat lisäävät valmiiksi omat yhdistimensä kaupalliseen ohjelmistoonsa ja että avoimen lähdekoodin ETL-työkalut tarjoavat niitä myös osana koko pakettia.

Seuraavassa Goldfedderin mainitsemassa poiminnan vaiheessa noudetut tiedot haetaan ja tallennetaan väliaikaiselle staging-alueelle, joka voi olla oma osansa tietovarastossa tai erillisiä tiedostotyyppejä, joihin saadaan tallennettua noudettu data. (Golfarelli & Rizzi, 2009) Joissain ETL työkaluissa tämä tilapäinen staging alue voi myös olla sisäänrakennettuna työkaluun.

Tässä vaiheessa täytyy pitää mielessä tietoturvallisuus, sillä staging datassa voi olla asiakkaisiin, tileihin tai muihin salattuihin tietoihin liittyviä henkilökohtaisia tietoja. Tällöin täytyy varmistaa, että prosessin aikana rajoitetaan arkaluontoisten tietojen määrää, käytetään salaustekniikoita ja varastoidaan väliaikaiset tiedot palomuurin taakse turvalliseen järjestelmään. (Goldfedder, 2020; Iyengar, 2002)

Viimeisessä poiminnan osa-alueessa varmistetaan saapuvan datan oikeellisuus. Vaikka koko datan louhinnan ja ETL prosessin aikana on tärkeää varmistaa datan oikeellisuus, on se kaikista tärkeintä poiminnassa, sillä korjaaminen tai menetetyn datan luominen seuraavien vaiheiden aikana luo suuren haasteen ETL kehittäjälle. (Goldfedder, 2020) Van der Lans (2012) lisää tähän, että, mitä lähempänä datan lähdettä data korjataan ja muokataan, sitä parempi lopputulos saadaan. Tämä johtuu siitä, että mitä lähempänä lopputaulua muutos tapahtuu, sitä raskaampaa se on. Van der Lans tiivistää tämän tärkeyden siten, että täydellisessä ympäristössä datan puhdistus ja muokkaus tapahtuisi suoraan niissä lähdejärjestelmissä, joissa se on mahdollista.

3.1.2 Muunto / Transform

ETL-prosessin toinen vaihe on tietojen muuntaminen. Siinä määritellään faktataulukoiden rakeisuus, mittataulukot, tietovaraston malli, johdetut faktat, hitaasti muuttuvat dimensiot ja faktattomat faktataulut. (El-Sappagh et al. 2011) Muunnosvaiheessa pyritään puhdistamaan ja muokkaamaan poiminnasta saatua dataa, jotta lopputulokseksi saataisiin dataa, joka on tarkkaa, oikeaa, virheetöntä, johdonmukaista ja yksiselitteistä. (Extract, transform, load -julkaisu)

Muunnosvaiheen pääasialliset tehtäviksi voidaan laskea Kakishin ja Kraftin (2012) mukaan

- Vain tiettyjen ladattavien sarakkeiden valitseminen
- Koodattujen arvojen kääntäminen (Taulukko 1)
- Vapaamuotoisten arvojen koodaus (Esim. Mies -> 1, Nainen -> 2)
- Laskettujen arvojen laskeminen (Esim. Tuotto)
- Lajittelu ja järjestäminen
- Tietojen yhdistäminen useista lähteistä
- Yhdistäminen (Esim. Summaus, keskiarvo)
- Surrogaattiavaimien arvojen luonti
- Transponointi/pivotointi
- Sarakkeen jakaminen useaan sarakkeeseen (esim. json purku, csv jakaminen)

Muokkauksille on olemassa käytännössä 2 vaihtoehtoa, ne voidaan korjata joko äsken mainituilla menetelmillä tai suoraan lähteeseen. Lähteessä suoritettava korjaus eli profilointi,

tarkoittaa lähdetietojen tarkistamista rakenteen, sisällön ja keskinäisten relaatioiden ymmärtämiseksi. (Goldfedder, 2020) Suoraan lähteeseen korjattaessa on hyvä huomioida monesti helposti huomattavissa olevat ongelmat:

1. Nimien ja lempinimien kirjoitusasut
2. Organisaatioiden oikeinkirjoitus ja lyhenteet
3. NULL-arvot kentissä, jotka on määritelty kohdetauluun NOT NULL ehdolla.
4. Aakkosnumeeriset tekstit numeroiden joukossa
5. Kirjoitusvirheet manuaalisesti syötetyissä arvoissa

Jos virheitä ei korjata lähteeseen, dataa käsiteltävässä kokonaisuudessa on jouduttava kehittämään omia sääntöjään, joilla virheet voidaan käsitellä yhdenmukaiseksi tai tietyn rakenteen omaaviksi. Merkittävin negatiivinen puoli luoduissa muunnossäännöissä on se, että niitä on pakko ylläpitää ja päivittää. (Goldfedder, 2020; Rahm & Do, 2000)

Taulukko 1. Eri kirjoitusasujen yhtenäistäminen

	Samaa tarkoittavat kentät			**Yhtenäistetty**
Yes	1	Y	K	1
No	0	N	E	0
Null	NaN	”		NULL
'LUT'	'LUT-University'	'Lappeenranta University of technology'		LUT

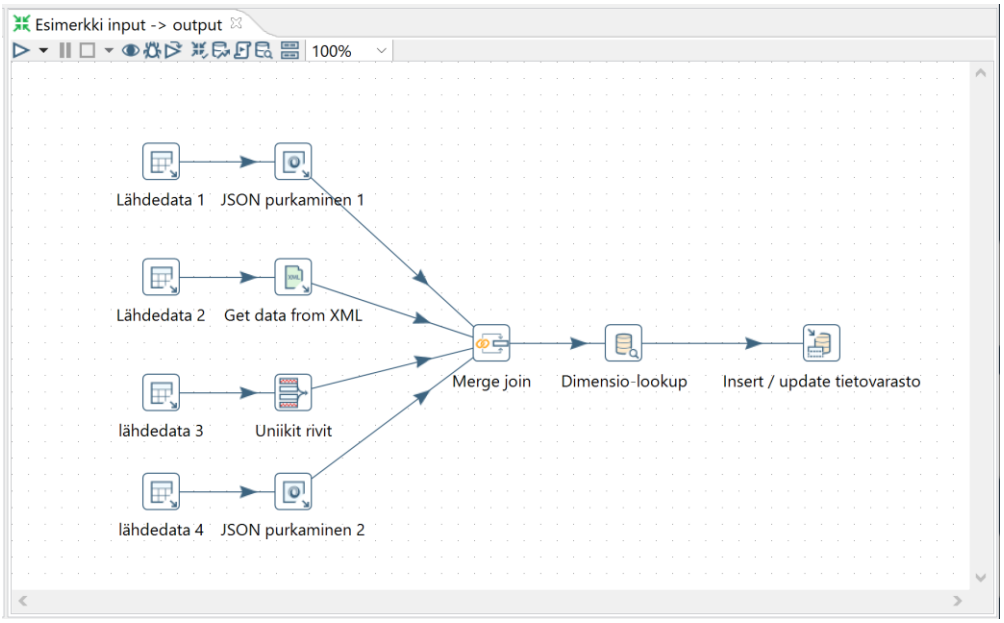
Kolmas vaihtoehto on olla tekemättä virheille mitään. Tässä tapauksessa data ei ole yhtenäistä ja esimerkiksi jos taulukossa 1 esiintyvä Lappeenranta University of Technology olisi lähtödatana, tulisi näille kaikille tekstikentilleen oma tekijä. Tällöin summaaminen tai suodattaminen yhden nimen perusteella ei toimisi, vaan jokainen kirjoitusasu olisi oma tekijänsä.

Tärkein asia datan muuntamisessa, kuten myös muissa datan siirtämisissä, on varmistaa, että muutos latautuu kohdetauluun oikein ja oikeassa muodossa. (Monica et al. 2019) Datan tarkistaminen käsittää kaikki muutokset, eli sarakkeiden ja rivien suodattamisen, hakuarvojen yhdistämisen, määreiden kuten kellonaikojen ja päivämäärien muuntamisen vakiomuotoon ja epäjohdonmukaisuuksien yhdistämisen samaan muotoon, kuten taulukkoon 1 on esitetty. (Goldfedder, 2020) Täten viimeisessä muutoksen vaiheessa, eli tuloksien vahvistamisessa on kyse tietojen muutoslogiikan oikeellisuuden vahvistamisesta.

3.1.3 Lataus / Load

Tietojen lataaminen on ETL-prosessin viimeinen vaihe. Tässä vaiheessa poimittu ja muunnettu data kirjoitetaan sekä dimensio- että faktatauluihin, joita loppukäyttäjät ja sovellusjärjestelmät pystyvät käyttämään. (El-Sappagh et al. 2011) Lataaminen voi tarkoittaa sitä, että ensin poistetaan ladatun aikavälin alkuperäinen data kohdetaulusta ja tämän jälkeen lisätään uusi data poistetun tilalle. Lataus voi myös tarkoittaa pelkästään uuden datan lisäämistä jo olemassa olevan datan päälle tai olemassa olevan datan päivittämistä. Lataamiseen voidaan myös lisätä kirjausketju (Audit trail), joka kertoo missä vaiheessa mikäkin data on lisätty tai päivitetty. (Schiefer & Jeng, 2003) Kirjausketju on yleensä juokseva luku, joka luodaan jokaiselle ajolle joka kerta kun se suoritetaan.

Goldfedderin mallissa latausvaihe, kuten poiminenkin, koostuu kolmesta alavaiheesta. Näistä ensimmäinen on molemmissa yhdistimien/liittimien asettaminen. Yhdistimien asettamisen periaatetta ei käsitellä uudelleen, mutta latausvaiheessa data liikkuu päinvastaiseen suuntaan, eli nyt lisäämme tai päivitämme tietoja kohdetauluun tai tauluihin. Näissä kahdessa vaiheessa suurimpana erona on nopeus. Joissain tapauksissa sisään tulevaa dataa saadaan useasta lähteestä, mutta ulospäin lähtevä data ladataan vain yhteen tauluun kuten kuvassa 7 on esitetty.



Kuva 7. Pentaho input -> output

Kuvassa vasemmalla olevat input stepit sisältävät dataa, jota käsitellään dataputkessa vähitellen. Merge Join-vaihe alkaa, kun edelliset vaiheet ovat valmiina, toisin sanoen kaikki poiminnat tehtynä. Täten Insert/Update käynnistyy, kun ensimmäinen rivi saapuu ja loppuu kun viimeinenkin rivi on käsitelty ja ladattu tietokantaan tai tiedostoon.

Lataamisen toisena vaiheena on päättää, miten käsitelty data ladataan. Kuten aikaisemmin sanottu, data voidaan ladata kokonaan tyhjään tauluun, eli uuteen tai tyhjennettyyn, lisätä inkrementaalisesti tai päivittää olemassa oleva data käsitellyllä datalla.

Viimeisenä vaiheena on peruutus tai palautus alkuperäiseen tilaan (Rollback). Tällä tarkoitetaan virheiden takia ETL-prosessin epäonnistumista ja tätä seuraavaa prosessia, millä käsitelty tai ladattu data saadaan poistettua osittain onnistuneesti tehdystä latauksesta. (Oliveira & Belo, 2017) Goldfedderin mielestä tämä on yleensä ETL-työkalun sisäisen ominaisuus, jotta kohdetauluun ei päädy osittain ladattua dataa. Rollback -tilanteessa dataa ei päädy prosessissa laisinkaan tauluihin vaan palataan alkuperäiseen tilaan.

3.2 Ralph Kimballin määritelmä

Ralph Kimballia voidaan pitää yhtenä tietovarastoinnin arkkitehteina. Kimball on kehittänyt malleja tietovarastoihin sekä dimensiomallinnukseen, joita voidaan pitää nykyään päätöksenteon tuen standardina.

Ralph Kimballin vuonna 2013 julkaisemassa kirjassa ***The-Data-Warehouse-Toolkit*** esitetään ETL-prosessin 4 pääasiallista komponenttia. Näistä ensimmäinen, poiminta tarkoittaa raakadatan keräämistä lähdejärjestelmistä ETL prosessin käsiteltäväksi. Seuraavassa vaiheessa eri lähteiden raakadata yhdistetään, puhdistetaan ja muokataan yhteiseen muotoon. Toiseksi viimeisessä vaiheessa, toimituksessa, luodaan datan rakenne ja ladataan tietovarastoon, josta sitä pystytään käyttämään. Viimeinen vaihe on hallinta, jossa nimensä mukaisesti hallitaan kaikkia ETL prosessiin liittyviä ja sen sisällä olevia järjestelmiä.

Ralph Kimballin vuonna 2004 kirjoittamassa tekstissä ETL prosessi on jaettu 38 alajärjestelmään, jotka ovat Kimballin mielestä tärkeimmät osiot toimivaan ETL kehittämiseen ja täten mahdollistavat sujuvan kehittämisen ja datan käytön.

Nämä 38 alajärjestelmää voidaan jakaa Danny Grassen ja Greg Nelsonin (2010) mukaan 6 eri osa-alueeseen. Ensimmäinen osa-alue on suunnittelu ja tietojen profilointi, johon luetaan vain **datan profilointijärjestelmä**. Datan profilointijärjestelmään lukeutuu sarakkeiden ominaisuusanalyysi, rakenneanalyysi, relaatiot sekä eri sääntöjen analyysit. Tässä osa-alueessa suunnitellaan tietovarastojen ja taulujen rakennetta pohjautuen datan todelliseen sisältöön. (Grasse & Nelson, 2010)

Toinen osa-alue, eli lähdetietojen poiminta pitää nimensä mukaisesti sisällään **poimintajärjestelmän**. Poimintajärjestelmään kuuluu adapterit lähdedatalle, datan järjestely ja suodatus jo lähteessä, tietotyyppimuutokset sekä data staging ETL prosessin aikana. Data staging tarkoittaa datan väliaikaista varastoimista.

Muutos ja lataus osa-alue on merkitty taulukkoon 2 vihreällä värillä, jotta osa-alueet saadaan helpommin ryhmiteltyä. Se pitää sisällään seuraavat osiot:

Taulukko 2. Muutos ja lataus (Grasse & Nelson, 2010)

5.	Datan sopeuttaja	Mukautettujen dimensioattribuuttien tunnistaminen useiden datalähteiden yhdistämiseksi
9.	Surrogaattiavainten luomisen järjestelmä	Mekanismi surrogaattiavainten tuottamiseksi riippumatta mistään muusta ulottuvuudesta
12.	Kiinteän hierarkian dimension rakennustyökalu	Tietojen pätevyyden tarkastus- ja ylläpitojärjestelmä kaikentyyppisille hierarkioille dimensioissa
13.	Muuttuvan hierarkian dimension rakennustyökalu	Tietojen pätevyyden tarkastus- ja ylläpitojärjestelmä kaikentyyppisille epämääräisille hierarkioille
14.	Monidimensioisen siltataulun rakentaja	Moni - moneen suhteita kuvaavan taulun luominen ja ylläpito
15.	Roskadimension rakentaja	Indikaattoreita ja merkkejä sisältävän dimensioiden luominen ja ylläpito
16.	Faktataulujen transaktioiden rakeisuuden lataaja	Järjestelmä tapahtumien rakeisuuden päivittämiseksi, mukaan lukien indeksien ja osioiden käsittely
17.	Jaksoittainen faktataulujen tilannekuvien lataaja	Järjestelmä jaksollisten rakeisten faktataulujen tilannekuvien päivittämiseen sisältäen indeksien ja partitioiden manipuloimisen
18.	Kerääntyvien faktataulujen tilannekuvien lataaja	Järjestelmä kertyvien faktataulujen tilannekuvien, dimensioiden ja määrien päivittämiseen sisältäen indeksien ja partitioiden manipuloimisen
19.	Surrogaatti avainten linja	Ohjattu monisäikeinen prosessi, jolla korvataan saapuvan datan luonnolliset avaimet tietovaraston surrogaattiavaimilla
20.	Myöhään saapuvien faktatietojen käsittelijä	Lisäys- ja päivityslogiikka viivästyneille faktatiedoille
21.	Ryhmittelyn rakentaja	Fyysisten tietokantarakenteiden luominen ja ylläpito kyselyn suorituskyvyn parantamiseksi. Sisältäen erilliset koontitaulukot ja toteutuneet näkymät
22.	Moniulotteisten kuutioiden rakentaja	Tähtiskeemapohjan luominen ja ylläpito moniulotteisten (OLAP) kuutioiden lataamista varten, dimensiohierarkioiden valmistelu
23.	Reaaliaikainen osioiden rakentaja	Logiikka jokaiselle kolmelle faktataulutyypille (alajärjestelmät 16, 17 ja 18), jotka ylläpitää "kuumaa osiota" muistissa, sisältäen vain tiedot, jotka ovat saapuneet staattisen tietovaraston viimeisimmän päivityksen jälkeen
24.	Dimensioiden hallintajärjestelmä	Hallintojärjestelmä dimensiohallinnalle, joka replikoi dimensiot keskitetystä sijainnista faktataulujentarjoajille
25.	Faktataulujen tarjoajajärjestelmä	Hallintajärjestelmä "faktataulujen tarjoajalle", joka vastaanottaa dimensiohallinnan lähettämät mukautetut dimensiot. Sisältää paikallisen avaimen korvaamisen, dimension versiotarkastuksen ja koontitaulukon muutosten hallinnan

Tämä osa-alue on ETL kehittäjän tärkein, sillä lähdedatasta muodostetaan analyyseissä ja koneoppimismalleissa hyödynnettävää mallinnettua dataa, jossa datalähteiden ja tyyppien integroiminen on avain onnistuneeseen tietovarastointiin (Grasse & Nelson, 2010)

Tiedonsiirron muutos on osa-alue, jossa valmistaudutaan ajan kanssa muuttuvaan lähdedataan. Muuttuneen tiedon käsittelyssä on kolme vaihtoehtoa. Korvata tai päivittää vanha data, kehittää logiikka, joka muuntaa vanhan datan uuteen muotoon tai tehdä uuden sarakkeen uuden tyyppiselle muuttuneelle datalle. (Grasse & Nelson, 2010)

Taulukko 3. Tiedonsiirron muutos (Grasse & Nelson, 2010)

2.	CDC (Change data capture) järjestelmä	esim. Lokitiedostojen lukijat, lähdepäivämäärän ja järjestysnumeron suodattimet
10.	SCD (Slowly Changing Dimension) prosessori	Muunnoslogiikka aikamuunnoksen käsittelemiseksi dimensioattribuutille : 1. päällekirjoitus, 2. luo uusi tietue, 3. luo uusi kenttä
11.	Myöhässä saapuvien dimensioiden käsittelijä	Lisäys- ja päivityslogiikka viivästyneille dimensiomuutoksille

Nämä kolme Kimballin alajärjestelmää on merkitty taulukkoon 3 punaisella värillä. Joskaan, ne ei tässä diplomityössä ole enää tarkemman tarkastelun alla, mutta on lisätty työhön niiden merkityksen vuoksi.

Laadun tarkistus, tarkastus ja poikkeusten käsittely/hallinta (Taulukko 4) voidaan jakaa automaattiseen ja manuaaliseen työhön. Laadun tarkistus ja tarkastus voidaan laskea ennalta tehtävään manuaaliseen työhön, jossa tarkastellaan seurauksia, jos datan laatu ei pysykään ennallaan. (Grasse & Nelson, 2010)

Taulukko 4. Laadun tarkistus (Grasse & Nelson. 2010)

4.	Datan puhdistusjärjestelmä	Tekstiparsinta, duplikaattipoistot. Lähdeavainten ja suhteiden säilytys
6.	Tarkastusdimension kokoonpanija	Metadatan rakentaminen niin, että se voidaan ladata faktatauluun yhtenä dimensiona
7.	Lähdedatan laadun käsittelijä	Datavirtojen testaus laadullisesta näkökulmasta
8.	Virhetapahtuman käsittelijä	Järjestelmä kaikkien ETL virhetapahtumien jakamiseen, raportoimiseen ja reagoimiseen

Automaattisesti tapahtuvaan tarkasteluun sisällytetään poikkeusten hallinta, koska myöhemmin esiteltävien ETL putkien ajastuksien muodostavat kaatumisraportit ja Kimballin ”Ongelman edistysjärjestelmä” tuo esille tapahtuneet ongelmat.

Integrointi tuotantoympäristön ja liiketoimintaprosessien komponenttien kanssa on viimeinen osa-alue ja se on merkitty listaan keltaisella (Taulukko 5).

Taulukko 5. Integrointi (Grasse & Nelson, 2010)

26.	Työn ajoitus -työkalu	Järjestelmä kaikkien ETL -töiden ajoittamiseen ja käynnistämiseen. Pystyy odottamaan monenlaisia järjestelmäolosuhteita, mukaan lukien riippuvuudet aiemmista töistä. Pystyy lähettämään hälytyksiä
27.	Workflow valvonta	Hallintapaneeli ja raportointijärjestelmä kaikille ajoille, jotka on ajastettu. Sisältää käsiteltyjen tietojen määrän, yhteenvedon virheistä ja tehdyistä asioista
28.	Palautus ja uudelleenkäynnistysjärjestelmä	Järjestelmä pysäytetyn työn jatkamiseksi tai koko työn peruuttamiseksi ja uudelleen käynnistämiseksi
29.	Rinnakkais-/putkilinjajärjestelmä	Yhteinen järjestelmä useiden prosessorien tai verkkopohjaisten laskentaresurssien hyödyntämiseksi ja jatkuvan datavirran toteuttamiseksi
30.	Ongelman edistysjärjestelmä	Automaattinen ja manuaalinen järjestelmä virhetilanteen nostamiseksi sopivalle tasolle korjausta ja seurantaa varten. Sisältää virhelokimerkinnät ja eri käyttäjien ilmoitukset
31.	Versionhallintajärjestelmä	Mahdollisuus arkistoida ja palauttaa kaikki ETL -putken metatiedot. Mahdollisuus vertailla versioiden eroja
32.	Versioiden siirtojärjestelmä	Siirrä ETL putken toteutus kehitysympäristöstä testaukseen ja sitten tuotantoon. Käyttöliittymä joka siirtää myös versionhallintaan
33.	Suku- ja riippuvuusanalysaattori	Näytä lopulliset fyysiset lähteet ja kaikki myöhemmät muunnokset valituista tietoelementteistä tai lopun tietoelementit ja raportin kentät, joihin mahdollinen muutos vaikuttaa
34.	Vaatimustenmukaisuuden toimittaja	Noudata säädöksiä todistaaksesi tärkeimpien raportoitujen tulosten alkuperä. Todista, että tietoja ja transformaatioita (putkia) ei ole muutettu. Mahdollisuus nähdä kuka on käyttänyt tai muuttanut tällaisia tietoja.
35.	Turvajärjestelmä	Hallitse roolipohjaista suojausta kaikista ETL-putkien tiedoista ja metatiedoista
36.	Varmuuskopiointijärjestelmä	Varmuuskopioi tiedot ja metatiedot palautusta, uudelleenkäynnistystä, suojausta ja vaatimuksia varten
37.	Metatietovaraston hallintajärjestelmä	Järjestelmä kaikkien ETL -metatietojen tallentamiseen ja ylläpitoon, mukaan lukien kaikki transformaatiologiikat
38.	Projektinhallintajärjestelmä	Järjestelmä ETL -kehityksen seurantaan

Tässä työssä käsitellään jo olemassa olevan järjestelmän kehittämistä koskien nimenomaan tätä osa-aluetta, sillä se pitää sisällään versionhallinnan sekä eri ympäristöjen välisen siirron ja metadatan tarkastelun. Nämä ovat pääkohteena tässä tutkimuksessa, ja niihin pyritään löytämään yhtenäistetty toimintatapa ja vähentämään kehittäjän mekaanista ja aikaa vievää työtä.

3.3 Testaus

ETL-testauksen päätavoitteena on todentaa ja lievittää tietovirheitä ja muita yleisiä virheitä, jotka ilmenevät ennen niiden käsittelyä analyysejä ja raportointia varten. (Yulianto, 2019) Tämän lisäksi testatessa voidaan testata eri kategorioita ja kattavasti koko ETL prosessi lähteestä lopputulokseen, rivimääriä lähteen ja kohdetaulun välillä sekä tietotyyppejä ja pituuksia toteutuksen ja tietomallin välillä. (Vandana & Sujatha, 2013)

Taulukko 6. ETL Testaustyypit (Yhdistetty: ETL Testing; Kautto, 1996; Yulianto, 2019)

Testaustyypit		
Ylätaso	**Testaus**	**Selite/Esimerkki**
Metadata	Datatyypit	Taulukon ja sarakkeen tietotyyppimääritelmät ovat tietomallin suunnittelumääritysten mukaiset
	Pituuden testaus	Sarakkeen tietomallimääritykset ovat pituudeltaan 100, mutta vastaava tietokantataulukon sarake on vain 80 merkkiä pitkä.
	Rajoitteiden testaus	Tarkistetaan, että tietomallin NOT NULL kentät on taulukossa NOT NULL
	Nimeämisstandardien testaus	Varmista, että tietokannan metatietojen, kuten taulukoiden, sarakkeiden ja hakemistojen, nimet vastaavat nimeämisstandardeja.
	Metatietojen tarkistus eri ympäristöissä	Vertaa taulukon ja sarakkeen metatietoja eri ympäristöissä varmistaaksesi, että muutokset on siirretty asianmukaisesti.
Datan eheys	Tietuemäärän vahvistus	Vertaa ensisijaisen lähdetaulun ja kohdetaulun tietueiden määrää. Tarkista hylätyt tietueet.
	Sarakkeen dataprofiilin vahvistus	Vertaa yksilöllisiä arvoja lähteen ja kohteen välillä, Vertaa maksimi, minimi, keskiarvo, maksimipituus, minimi pituus
	Vertaa koko lähde- ja kohdetietoja	
Datan laatu	Duplikaattien tarkistukset	Etsi duplikaattirivejä, joilla on sama yksilöllinen avain tai yksilöllinen sarakeyhdistelmä liiketoimintavaatimuksen mukaan.
	Tietojen validointisäännöt	Tarkista onko tiedot järkeviä. Esim. syntymäpäivä tulevaisuudessa tai 1700- luvulla ihmisellä, joka elää
	Tietojen eheystarkastukset	Laske tietueiden määrä, joissa on null avainarvoja alitaulukossa, Virheellisten FK arvojen lukumäärä taulukossa, joilla ei ole vastaavaa PK avainta ylätason taulukossa
Tietojen muuntamistestaus	Lasilaatikkotestaus (White Box)	White box testauksessa tapaukset johdetaan suoritettavan ohjelman jokaiseen haaraan, jotta jokainen polku tai tulosvaihtoehto testataan
	Black Box	Muuntamisen toimintaa tarkastellaan input-output käyttäytymisen avulla

Reeves (2009) lisää kirjassaan ***A Manager's Guide to Data Warehousing***, että ETL testausta tehdään liian vähän, ja että mitä aikaisemmin testaaminen tehdään, sitä helpompaa se on. Tämän lisäksi Reeves listaa dataputken koodin testaamisen kuusi testaustyyppiä, jotka ovat:

1. Yksikkötestaus

2. Regressiotestaus

3. Laadunvarmistustestaus

4. Hyväksymistestaus

5. Suorituskyvyn testaus

6. Integraatiotestaus

Yksikkötestauksen tekee kehittäjät, jotka testaavat tässä testaustyypissä sitä, että jokainen kokonaisuuden osista toimii, jokainen toiminto toimii suunnitellusti ja että jokainen muutos toimii halulla tavalla. Näitä muokkauksia on listattu juuri taulukkoon 6. **Regressiotestauksessa** varmistetaan, että tehdyt muutokset eivät ole vaikuttaneet olemassa olevien dataputkien tai ominaisuuksien toimintaan, joita ei ole tarkoitus muuttaa. (Wong et al. 1997) **Laadunvarmistustestauksessa** ETL kokonaisuus viedään testiympäristöön, jossa sen toiminnallisuutta testataan ajamalla se läpi ja tarkastelemalla tulosta. (Reeves, 2009) **Hyväksymistestauksesta** puhuessa kokonaisuuden tilaaja testaa, että tilattu kokonaisuus toimii kuten oli tarkoitus ja kuten on määritelty. Tätä voidaan pitää asiakkaan versiona regressiotestauksesta. (Reeves, 2009; Miller & Roy, 2002) **Suorituskyvyn testauksesta** puhuessa on selvää, että sillä tarkoitetaan prosessin suorituskykyä. Näitä on esimerkiksi nopeus (Elgamal et al. 2013) ja prosessin käyttämät resurssit. Tämän lisäksi prosessille tehdään rasitustestausta suuremmalla datamäärällä ja tarkkaillaan suorituskykyä. (Reeves, 2009) **Integraatiotestauksessa** ETL testataan ajamalla se ensimmäistä kertaa. Testattavia ominaisuuksia on poiminnan osalta kaikki kolme menetelmää, eli alkulataus, historialataus ja jatkuva lataus. (Reeves, 2009) Sen lisäksi tarkastellaan datavirtaa ja sitä, toimiiko jokainen toiminnallisuus oikein. Työssä esitellään testausta, sillä se on ETL kehityksen kannalta tärkeässä roolissa. Samaten testausta tehdään iteratiivisesti useassa ympäristössä, jolloin tarvitaan ympäristöjen välisiä siirtoja.

3.4 ETL työkalut

ETL kehittämistä voidaan tehdä ETL-työkaluilla tai ohjelmointikielellä koodaamalla. ETL työkalun pitäisi olla kykenevä aikaisemmin mainittuihin ETL prosessin vaiheisiin. Aikaisemmin ETL kehitystä tehtiin koodaamalla, mutta nykyään työkalut ovat kehittyneet siten, että niissä on graafinen käyttöliittymä, ne voi olla pilvessä ja ne tukevat metadatan keräämistä. (Patel & Patel, 2020) ETL työkalun ominaisuudet siis antavat selkeän edun

kehittämiseen, mutta työkalun vaihtaminen voi luoda ongelmia, sillä työkalujen välillä ei ole standardeja dataputkien rakenteissa tai järjestyksissä. (Gupta, 2014)

3.4.1 Käsin koodaamisen hyödyt ja haitat

Ennen ETL-työkaluja, kehittäjät käyttivät kustomoituja koodeja ETL-toimintojen suorittamiseen. Tällä menetelmällä tehdyt ohjelmat olivat pitkiä ja vaikeasti dokumentoitavia. Madhu Zode (2007) on ETL kehitystä tutkineessa julkaisussaan maininnut, että aikaisemmin ETL-kehittäjän oli usein käytettävä koko prosessin suorittamiseen eri ohjelmointikieliä, kuten:

1. Perl-skriptejä tietojen poimimiseen lähdejärjestelmistä ja muunnosten suorittamiseen

2. SQL Loader- ja PL/SQL Bulk proseduuria käytettiin tietojen lataamiseen

3. Shell skriptejä, käytettiin siihen, että saatiin ETL kokonaisuus pakattua suoritettaviksi ohjelmiksi

Perl-skriptejä käytettiin muunnosten suorittamisessa, sillä esimerkiksi tekstinkäsittely, datan käsittely sekä analyysi on tällä ohjelmointikielellä helppoja. Perl itsessään on Larry Wallin kehittämä proseduraalinen ohjelmointikieli. Shell skriptejä käytettiin käynnistämään aiemmin mainitut Perl skriptit sekä SQL koodit. Nämä skriptit toimivat siis kuorina, joka käynnistettiin ja pitivät sisällään koodin, jolla suoritettiin poimiminen, muunto ja lataus. Shell skriptien tiedostotunniste on .sh

Goldfedder (2020) ja Zode (2007) ovat molemmat listanneet käsin koodaamisen hyötyjä ja haittoja. Molempien kirjoittajien listoista yhteenvetona voidaan sanoa, että käsin koodaamisen hyödyiksi voidaan laskea sovelluksien riippuvuuden poistumisen. Tämä siksi, koska sovelluksien oppimiskäyrä voi olla hyvinkin jyrkkä. Käsin koodatessa ETL putket saadaan tehtyä juuri halutunlaisiksi, ne ovat ketterämpiä, monipuolisempia ja ne kykenevät kaikkeen sekä metadatan tallennus on tarkempaa ja suoraviivaisempaa. Käsin koodatessa myös testaus on helpompaa, koska pystytään käyttämään ohjelmistotuotannosta tuttuja automatisoituja yksikkötestaustyökaluja. Viimeisenä, tosin epäsuorana hyötynä kirjoittajat sanoo olevan lyhyen aikavälin taloudelliset säästöt verrattuna lisenssejä vaativiin ohjelmiin.

Vaikka käsin koodaamisessa on useampi hyöty, on Goldfedder ja Zode listanneet myös haitat.

- Muutosten dokumentointi täysin kehittäjän vastuulla

- Koodin kommentoinnin tärkeys, jos kehittäjiä on useita. Samalla vaatii kaikilta kehittäjiltä hyvää ymmärrystä ja osaamista käytettävistä kielistä.

- Käsin koodattuja ETL putkia on muokattava jatkuvasti ja monissa tapauksissa jopa kirjoittaa kokonaan uudelleen

- ETL:n käsin koodaus edellyttää metatietotaulukoiden ylläpitämistä erikseen. Kaikki uudet muutokset edellyttävät muutoksia myös metatietotaulukoihin

- Käsin koodattujen ETL-ohjelmien suoritusnopeus on todennäköisesti hitaampi, koska käsin luodut ohjelmat ovat tyypillisesti yksisäikeisiä, kun taas nykyaikaiset ETL-työkalut luovat monisäikeisiä ETL kokonaisuuksia.

3.4.2 ETL työkalut, hyödyt ja haitat

Nykyään saatavilla olevat ETL-työkalut sisältävät runsaasti muunnosominaisuuksia. Yleensä ne tukevat useita tietokantoja ja tiedostotyyppejä, moniulotteisia malleja, surrogaattiavainten luomisen, erilaisia muunnostoimintoja ja natiivia tietokantaa. (Zode, 2007) Niissä voi olla sisäisiä metadatan varastoja, jotka voivat olla erilaisia kuin tietovaraston oma metadatavarasto. Nykyään monet työkalut tarjoavat myös helposti käytettävän käyttöliittymän, jonka avulla kehittäjä voi työskennellä ilman pitkää harjoittelua. Näin ollen oppimiskäyrääkään ei ole jyrkkä, joka voitiin käsin koodaamisen hyödyksi laskea. ETL työkaluissa on myös ominaisuuksia ETL prosessien hallintaan, kuten valvonta ja aikataulutus, jotka ovat Kimballin hyvän järjestelmän ratkaisevassa osassa.

Kuten käsin koodaamisen hyödyt, on myös ETL työkalujen vahvuudet koostettu Goldfedderin ja Zoden julkaisuista.

1. Nykyaikaiset ETL -työkalut pystyvät käsittelemään monimutkaisimmat muunnokset nopeammin

2. Työkalujen tuottama koodi voidaan ajaa eri alustoilla erittäin nopeasti. Tämä mahdollistaa myös kuorman jakamisen useille alustoille suorituskyvyn optimoimiseksi
3. Tietojen rinnakkaisuus, komponenttien rinnakkaisuus ja ETL putkien rinnakkaisuus, jotka johtavat nopeampiin suoritusaikoihin
4. Työkaluissa on ajoitustoiminto, joilla ajastaa ETL prosessit tiettyihin aikoihin
5. Tukee kaikkia tietovarastojen käsitteitä, kuten normalisointia, denormalisointia, hitaasti muuttuvaa dimensiota
6. Nämä työkalut tarjoavat myös versionhallinnan ominaisuuden jollain tasolla
7. Debuggeri -tuki
8. Automaattinen dokumentointi, graafinen käyttöliittymä tuottaa helpomman ja selkeämmän dokumentin kun käsin koodattu ja kommentoitu koodi.
9. Metatietojen perusta ETL-prosessin kaikkiin vaiheisiin
10. Versionhallinta usean kehittäjän ympäristöihin sekä yhdenmukaisten versioiden varmuuskopiointi ja palauttaminen (Ei kuitenkaan toimi eri ympäristöjen välillä)
11. Muunnoslogiikka, kuten sumea sovitusalgoritmi, integroitu pääsy nimen ja osoitteen deduplikointirutiiniin ja tiedonlouhinta-algoritmit
12. Parempi suorituskyky kehittäjille, joilla on matalampi asiantuntemus
13. Käsittelyominaisuudet, mukaan lukien tehtävien automaattinen rinnastaminen ja automaattinen vikasieto, kun käsittelyresurssi ei ole käytettävissä
14. Kehittyneet käsittelyominaisuudet, mukaan lukien tehtävien automaattinen rinnastaminen ja automaattinen vikasieto, kun käsittelyresurssi ei ole käytettävissä

Toisaalta ETL työkalutkaan eivät ole täydellisiä ja näiden huonoiksi puoliksi tai rajoitteiksi voidaan laskea ainakin sen, että nykyään saatavilla olevat työkalut eivät riitä tukemaan reaaliaikaisia sovelluksia eivätkä kata useita lähdetietokantoja. Lisäksi monet viimeaikaiset ETL-työkalut eivät tue integrointia metatietotasolla loppukäyttäjän työkalujen kanssa.

3.5 ETL ympäristöt

Liiketoiminnan kannalta tärkeiden sovelluksien kehittäminen ei tapahdu satunnaisesti kehittäjän tietokoneella, vaan yleisesti kehitykseen liittyy kehitysprosessi, joka kattaa itse kehityksen, testauksen ja toimituksen. (Moss & Atre, 2003) ETL kehityksessä, kuten muussakin ohjelmistokehityksessä eristettyjä ympäristö tulee olla useampi. Useamman ympäristön merkitys kasvaa, mikäli kehitettävässä koodissa on jotain väärin ja lopputulos esimerkiksi pyyhkii osan koko tietokannasta mennessään. (Nayem et al. 2016) Näitä ympäristöjä on yleensä kolme, mutta määrää ei ole rajoitettu eli ympäristöjä voi olla niin paljon, kun organisaatio haluaa. Kolme yleisintä ympäristöä ovat kehitysympäristö, testausympäristö ja tuotantoympäristö. Näiden nimet voivat olla erilaiset, mutta tarve ja tarkoitus on kuitenkin sama. (Nayem et al. 2016; Goldfedder, 2020)

3.5.1 Kehitysympäristö

Kehitys (Development / Dev) on jokaisen kehittäjän omalla tietokoneella sijaitseva ympäristö, jossa he päivittävät koodia. Kehitysympäristö toimii siis ympäristönä, jossa kehittäjä kehittää uusia ja muokkaa olemassa olevia ETL kokonaisuuksia. Tämä ympäristö muuttuu koko ajan ja nopeasti, joten sitä kutsutaan joskus "sandbox"-termillä. (Goldfedder, 2020) Kehitysympäristön avulla tiimi voi tehdä erilaisia kokeiluja ja testejä, ja sen tulisi olla yhteydessä simuloituun dataan tai testaustietokantoihin, joissa ei ole tuotantodataa. Kehitysympäristöjen avulla uusien kokonaisuuksien tai muokkauksien myötä dataan tulevat merkittävät häiriöt minimoidaan ja ne vaikuttavat vain testaaviin kehittäjiin. (Nayem et al. 2016; Goldfedder, 2020). Kehitysympäristön lisäksi isommilla yrityksillä on rinnallaan niin sanottu prototyyppiympäristö, jossa tehdään proof-of-concept ja demoprototyyppejä. Pienemmissä organisaatioissa nämä kaksi on kuitenkin sulautettu yhteen eli kehitysympäristöön. (Moss & Atre, 2003)

3.5.2 Testiympäristö

Kehitysympäristöstä koodi siirretään testiympäristöön (Staging/Test), jonka tarkoituksena on nimensä mukaisesti mahdollistaa suurin osa arvioinnista ja laadunvarmistustestauksesta. Siksi ideaalitilanteessa pyritään siihen, että testiympäristö olisi mahdollisimman tarkka kopio tuotantoympäristöstä vähintään testauksessa käytettävän datan osalta (Sherman, 2014). Tällä tarkoitetaan mahdollisimman tarkkaa kopiota datan rakenteesta sekä lähde- että kohdejärjestelmässä. Moss & Atre (2003) lisää, että mikäli ympäristöt on määritelty eri tavalla, voi ohjelman siirtämisellä ympäristöstä toiseen olla suuri vaikutus sen toimintaan. Tietoturva tai muista teknisistä syistä ei ole mahdollista tallentaa datan tarkkoja kopioita, mutta mitä lähempänä tuotantorakennetta ollaan, sitä valmiimpana tuotantoon viennin koittaessa ollaan. (Goldfedder, 2020)

Goldfedderin mukaan testausympäristön mahdollisimman hyvä tilanne varmistaa, että kehittäjän on mahdollista testata kehittämäänsä dataputkea eristyksissä. Näin ollen kehittäjä tietää, että kaikki kohdatut ongelmat voidaan ratkaista iteroinnin ja huolellisen tarkistuksen avulla. Mikäli testaus tehdään tuotantoympäristössä, joudutaan pahimmassa tapauksessa tärkeät tuotantotiedot hakea päivittämällä tai palauttamalla varmuuskopioista, joka voi olla pitkäkin prosessi.

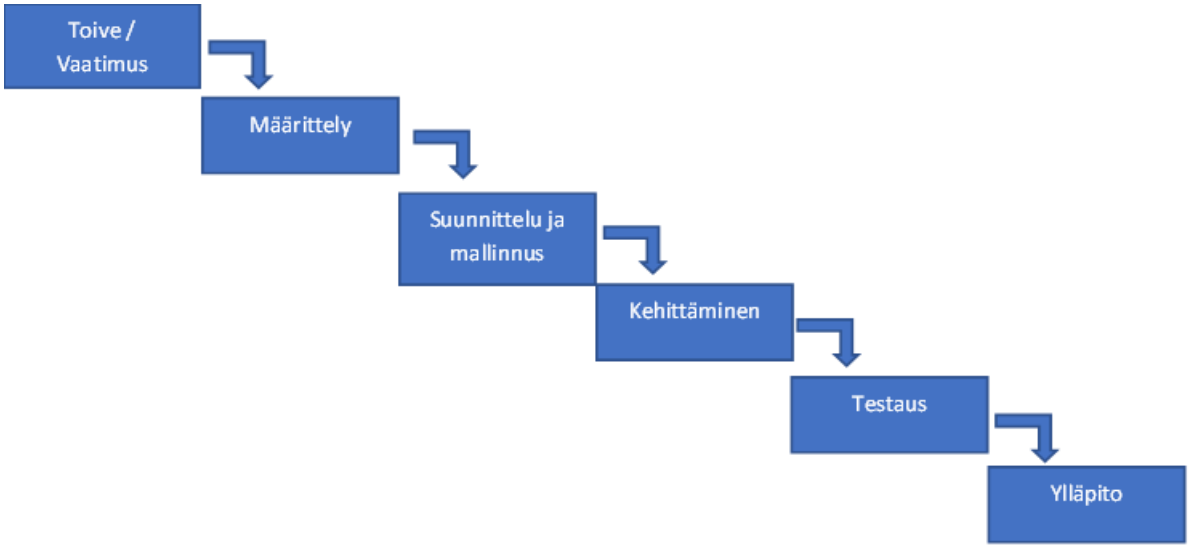
3.5.3 Tuotantoympäristö

Tuotantoympäristö (Production / Prod) on kehitetyn tai muokatun ETL kokonaisuuden viimeinen ympäristö, ennen seuraavaa muokkausta ja jatkokehitystä. Tuotantoympäristö on ympäristö, jossa ETL koodi pyörii ajastuksilla kellon ympäri ja joka on liitettynä tuotantodataan. Tuotantoympäristöön tulisi koskea vain silloin, kun kehitetty ETL kokonaisuus on käynyt testiympäristössä testattavana ja on valmis tuotantoon vientiin. (Goldfedder, 2020; Sherman, 2014) Tuotantoympäristön tietoturva on paljon tarkempi ja tiukempi kuin muissa ympäristöissä. Tietoturva on tiukempi, sillä tuotantoympäristö sisältää kaiken arkaluontoisimmankin datan, jota muissa ympäristöissä ei ole. (Moss & Atre, 2003) Näiden kaikkien ympäristöjen kokonaisuus varmistaa sen, että tuotantoympäristö on luotettava sekä datan että toiminnallisuuden puolesta. (Reeves, 2009) Reeves lisää tuotantoympäristölle

tärkeimmiksi ominaisuuksiksi Ralph Kimballin listaamia ominaisuuksia. Näitä on lähdedatassa saapuvien poikkeuksien käsittely, ilmoituksien ja virheviestien lähettäminen tarpeen vaatiessa, seurantaketjun muodostamisen, sekä varmuuskopioinnin itsestään.

3.6 ETL Kehityksen prosessi

ETL kehityksen prosessi noudattelee pääpiirteittäin samanlaista prosessia kuin ohjelmistokehityksenkin. Kuva 8 esittelee yksinkertaisen vesiputouskaavion, jossa kuvataan kehityksen vaiheita alkutarpeesta lopulliseen tilaan, jota voidaan pitää ylläpitona.



Kuva 8. ETL kehityksen vesiputousmalli (Yhdistetty Bassil, 2012; Mishra & Dubey, 2013)

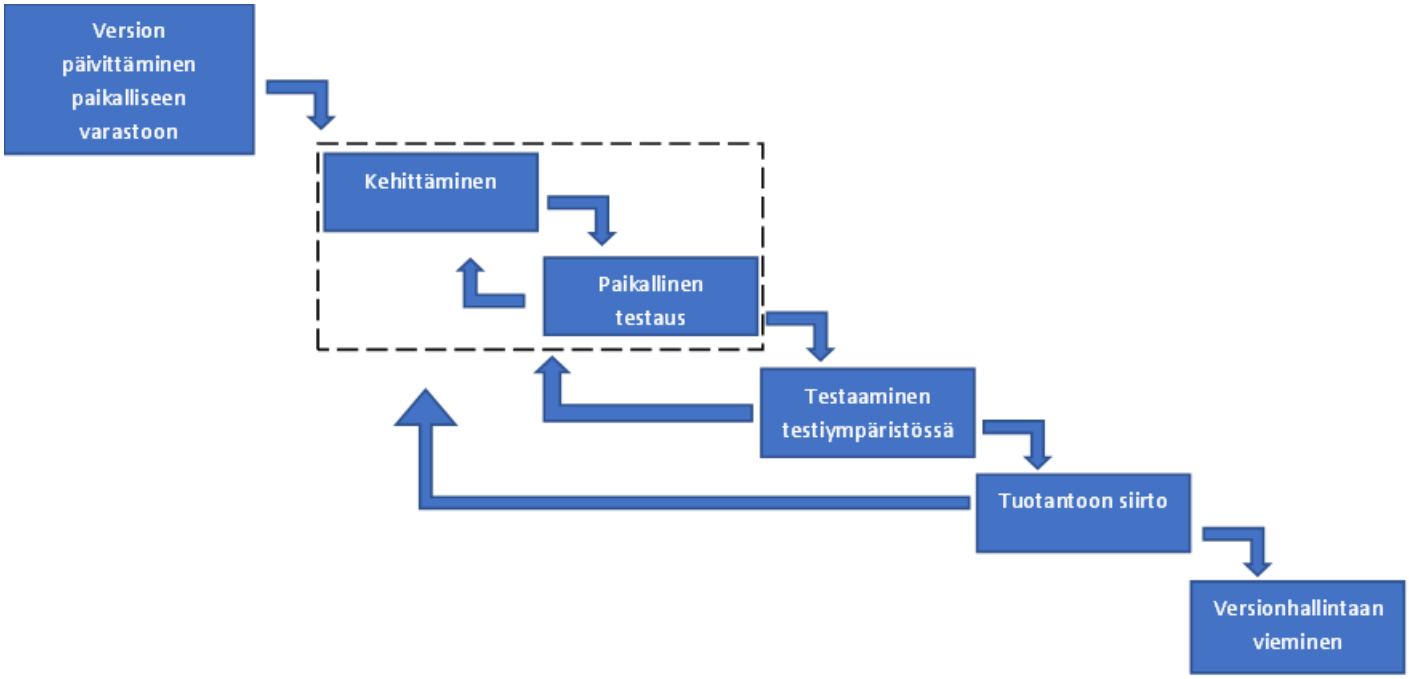
Toive tai vaatimus ETL muutoksesta saadaan datan loppukäyttäjältä tai omistajalta. Nämä voivat koskea kokonaan uutta dataa ja ETL kokonaisuuksia tai jo olemassa olevan muunnosprosessin muokkaamista. Olemassa olevan ETL kokonaisuuden muokkauksessa lopputuloksena syntyvään tauluun saadaan data eri muodossa, eri tavalla ryhmiteltynä tai kokonaan uusilla ehdoilla. Määrittelyvaiheessa määritellään mitä oikeastaan halutaan ja miten se on saavutettavissa. (Bassil, 2012) Tästä päästään suunnittelun ja mallinnuksen vaiheeseen, jossa dokumentoidaan kuinka haluttu muutos tai kehitys tulisi tehdä. Viimeisenä kuvaajassa nähdään ylläpito, joka tarkoittaa sitä, että ETL kokonaisuus on ajastettuna tuotantoympäristössä. Tässä vaiheessa sitä valvotaan sekä mahdollisesti muokataan tulevaisuudessa, mikäli

määrittely muuttuu tai havaitsematta jääneitä virheitä on jäänyt koodiin. (Mishra & Dubey, 2013)

Todellisuudessa jokaiseen vaiheeseen kuuluu laajempi kokonaisuus alavaiheita, mutta tässä työssä keskitytään vaan kehitys, testaus sekä ylläpitovaiheisiin sillä ne ovat työn aihepiirin kannalta oleellisessa osassa.

ETL-kehittäjän varsinainen työ alkaa vasta, kun suunnittelu ja mallinnus on saatu maaliin. Tässä vaiheessa tulevan taulun relaatiot on mallinnettu ja lopputuloksena olevan taulun kentät on määritelty niin, että tiedetään mistä ne saadaan ja mihin muotoon ne muokataan. Jos kyseessä on olemassa olevan kokonaisuuden kehittäminen, on ETL kehittäjän ensimmäinen vaihe paikallisen version päivittäminen versionhallinnasta tuoreimpaan versioon. Tämän jälkeen kehittäjä alkaa kehittämään ETL kokonaisuutta malliin kirjatulla tavalla. Kun kehittäminen on saatu valmiiksi, kokonaisuus viedään testiympäristöön, jossa kokonaisuus testataan ja tarkastetaan. (Mishra & Dubey, 2013) Mikäli kokonaisuus on mallin mukainen ja toimii oikein, viedään tuo kokonaisuus testiympäristöstä tuotantoympäristöön ja yleensä tarkistetaan toimivuus erikseen tuotantodatalla. Ylläpitovaiheessa kehitetty kokonaisuus jätetään ajastettuna pyörimään tuotantoympäristöön ja viedään versionhallintaan, josta muut kehittäjät päivittävät sen omaan paikalliseen varastoon.

Kuten Mishra & Dubey (2013) mainitsee, on todellisuudessa kehittäminen ja testaus iteratiivinen prosessi ja testausta tehdään jo kehittäjän toimesta kehittäjän omassa kehitysympäristössä. Kuvassa 9 nähdään heidän kaavioitansa mukaillen tarkemmin tehty vesiputouskaavio kehittämisen, testauksen ja ylläpidon osalta.



Kuva 9. Tarkempi kuva kehitysprosessista (mukaillen Mishra & Dubey, 2013)

Ajoittain, korjauksen ollessa helppo tai kun on huomattu aikeisemmin kehitetyn kokonaisuuden lopputuloksen olevan erilainen kun toivottu, voidaan muutos tai korjaus tehdä suoraan tuotantoympäristöön. Tätä kutsutaan hotfixiksi, jossa kehitys tehdään suoraan tuotantoympäristössä olevaan versioon jättäen välistä kaikki kehittämisen ja testauksen muut vaiheet. Tässäkin tapauksessa kuitenkin päivitetty versio pitää viedä versionhallintaan muita kehittäjiä varten ja varmuuskopioksi.

4 Versionhallinnan tarkastelu

Versionhallinnan järjestelmällä tarkoitetaan erillistä järjestelmää, jonka tehtävänä on tallentaa vanhat versiot ja kehitettyjen versioiden muutokset. Versionhallintaa käytetään, jotta jälkeenpäin on mahdollista tarkastella tehtyjä muutoksia ja palauttaa versio mistä kehityksen vaiheesta tahansa. (Somasundaram, 2013). Versionhallinta on erityisesti kannattavaa, kun projektissa työskentelee useampi kuin yksi kehittäjä, mutta jo yhden kehittäjän projektissa versionhallinta tuo hyötyjä. (Mäkiaho et al. 2014)

Sekä ohjelmistotuotannossa, että ETL kehityksessä on huomattu, että ennen lopullista version syntymistä testataan useaa alustavaa versiota. Alustavien versioiden määrä kasvaa, kun kehitetään suurempia ja monimutkaisempia järjestelmiä, joka johtaa koodin ja tiedoston vaikeaan hallintaan ja järjestelyyn. (Zolkifli et al. 2018) Tästä syystä versionhallinnan olemassaolo todella auttaa kehittäjiä nopeuttamaan ja yksinkertaistamaan kehitysprosessia. Ilman versionhallintaa kehittäjien tulisi pitää omilla kovalevyillään useita versioita yhdestä kehitettävästä kokonaisuudesta. Tämä altistaa riskille, että kehittäjä alkaa kehittämään väärää versiota tai poistaa uusimman version, mikä voi johtaa tehdyn työn menettämiseen.

Versionhallinnan historiassa voidaan sanoa olevan kolme sukupolvea, jotka on esitetty taulukossa 7. Ensimmäisen sukupolven järjestelmiä voidaan kutsua paikalliseksi versionhallinnaksi ja näitä on ollut esimerkiksi Source Code Control System (SCCS), joka kehitettiin vuonna 1972 ja RCS (Revision Control System) joka julkaistiin 1982. Näissä versionhallinta toimii kehittäjän omalla tietokoneella ja versioiden väliset muutokset kirjataan vain kehittäjän omalle tietokoneelle. Ensimmäisen sukupolven järjestelmissä oli kuitenkin suuri ongelma yhteisen kehittämisen hallinnassa, sillä kehitystä pystyi tekemään vain yksi kehittäjä kerrallaan. (Mäkiaho et al. 2014)

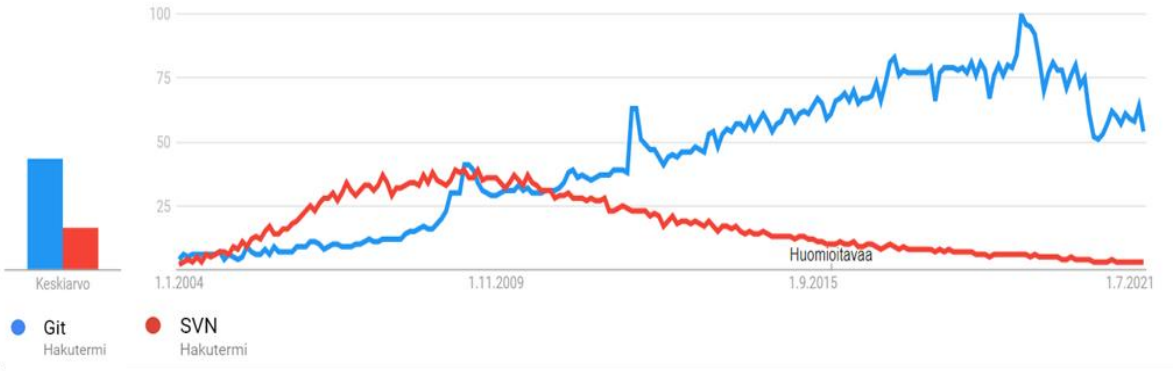
Toisen sukupolven järjestelmistä voidaan käyttää nimitystä keskitetyt järjestelmät, jossa yhteinen kehittäminen otettiin paremmin huomioon ja yhtä tiedostoa pystyi kehittämään useampi kehittäjä. Tämä tehtiin mahdolliseksi yhdistämällä eri versioiden muutokset ja vasta sitten versionhallinnan keskitettyyn varastoon viemisellä.

Hajautettua versionhallinnan järjestelmää pidetään kolmantena sukupolvena, joka mahdollistaa kehittämisen myös offline-tilassa. Alla olevassa taulukossa 7 on lyhyesti esitetty eri sukupolvien erot ja esimerkit.

Taulukko 7. Versionhallinnan sukupolvet ja ominaisuudet (Raymond, 2011)

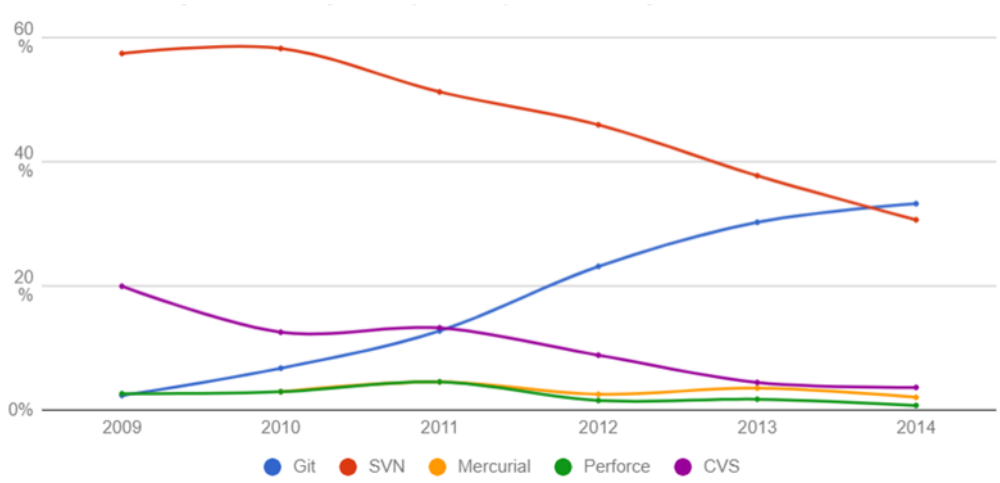
Sukupolvi	Verkosto	Toiminnot	Rinnakkaisuus	Esimerkit
Ensimmäinen	Ei	Yksi tiedosto kerrallaan	Lukot	RCS, SCCS
Toinen	Keskitetty	Monitiedosto	Yhdistäminen ennen kommitointia	CVS, SourceSafe, Subversion, Team Foundation Server
Kolmas	Hajautettu	Muutosjoukot	Kommitointi ennen yhdistämistä	Bazaar, Git, Mercurial

Kuva 10 esittää Googlen hakutermien statistiikkaa, jossa on vertailtu 2. ja 3. sukupolven suosituimpien järjestelmien nimiä, eli SVN ja GIT.



Kuva 10. Versionhallinnan työkalujen suosio Googlen hauissa (Google Trends)

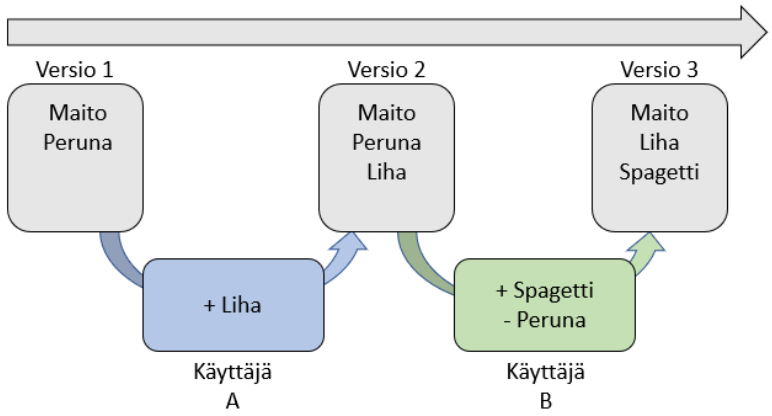
Kuten kuvassa 10 nähdään ovat 3. sukupolven järjestelmä ohittanut toisen sukupolven järjestelmän suosiossaan ja ero kasvaa entisestään. Parhaimman kuvan eri järjestelmien suosiosta saadaan kuitenkin Eclipse Communityn joka vuosi vuoteen 2014 asti tehdystä kyselystä, jossa vertaillaan versionhallinnan suosiota käyttäjämäärittäin (Kuva 11). Tässä kuvassa data loppuu vuoteen 2014, mutta siitä huomataan kuitenkin SVN:n laskeva trendi sekä GITin yleistyminen käytössä.



Kuva 11. Eclipse Communityn versionhallintajärjestelmien käyttö (RhodeCode, 2016)

4.1 Keskitetyt järjestelmät

Nykyään yrityksissä voidaan sanoa olevan versionhallinnan osalta kaksi erilaista lähestymistapaa, jotka ovat 2. ja 3. sukupolven järjestelmät eli keskitetty versionhallintajärjestelmä (CVCS) ja hajautettu versionhallintajärjestelmä (DVCS). Keskitetyssä järjestelmässä on olemassa nimensä mukaisesti vain yksi keskitetty varasto, joka sijaitsee serverillä. Näin ollen jokainen, joka haluaa kehittää jotain tiedostoa tai kokonaisuutta, on oltava verkossa. Tässä on kuitenkin ongelma, koska mikäli yhteys ei toimi, on kehittäjällä käytössään vaan viimeinen versio, jotka on haettu yhteyden toimiessa. Tällaisessa tilanteessa kehittäjä ei myöskään pysty julkaisemaan omaa kehitettyä versiota keskitettyyn varastoon. Tämän lisäksi pahimmassa tilanteessa serverillä oleva ainut varasto korruptoituu ja kaikki versionhallinta ja tiedostot menetetään. (Somasundaram, 2013)



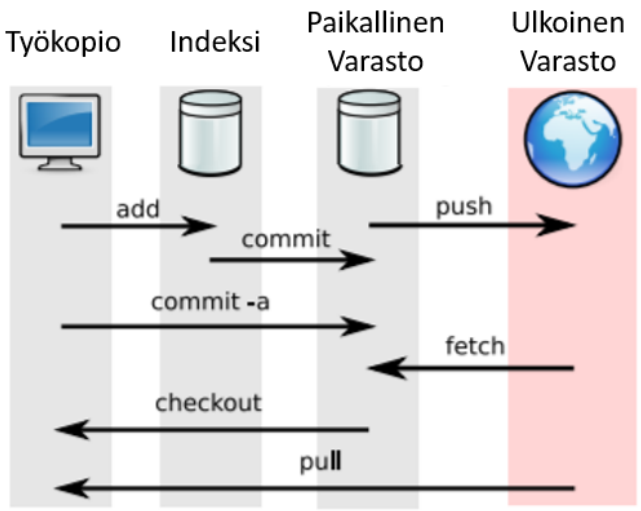
Kuva 12. Keskitetty versionhallintajärjestelmä (mukaillen Malmsten, 2010)

Kuvassa 12 on kuvattu yksinkertaistetusti, kuinka keskitetty järjestelmä toimii. Kuvassa huomataan harmaalla pohjalla keskitetty varasto, josta haetaan viimeinen tallennettu versio ja jonne tallennetaan kehitettäessä muuttunut versio. Optimaalisessa tilanteessa käyttäjä A hakee työkopion serverillä sijaitsevasta varastosta, tekee muutokset ja tallentaa kehitetyt version serverille, josta käyttäjä B hakee tämän kehitetyn version ja jatkaa kehittämistä. Kuvassa sininen ja vihreä ovat kehittäjien muutoksia keskitetyn varaston versioon.

4.2 Hajautetut järjestelmät

Hajautetut järjestelmät kehitettiin, jotta keskitettyjen järjestelmien huonoista puolista päästäisiin eroon. Tässä järjestelmässä jokaisella kehittäjällä on omalla koneellaan paikallinen varasto, joka kommunikoi serverillä olevan varaston kanssa. (Malmsten, 2010; Zolkifli et al. 2018) Näin ollen kehitystä tehdessä ei ole merkitystä onko tietokone yhteydessä serverillä olevaan varastoon, koska työkopiot tallennetaan ensin omaan paikalliseen varastoon ja sieltä viedään serverille.

Hajautetun mallin kehityksessä yksinkertainen toimintamalli on seuraavanlainen. Ulkoisesta varastosta vedetään tuoreimmat tiedostot paikalliseen varastoon, josta ne siirretään työkopioon. Työkopiossa tiedostoon tehdään muutoksia ja muokattu tiedosto siirretään takaisin paikalliseen varastoon. Viimeisenä paikallisessa varastossa oleva kehitetty tiedosto viedään ulkoiseen varastoon, josta muut kehittäjät saavat muutetun tiedoston päivittäessään paikallisen varastonsa. (Teittinen, 2019)

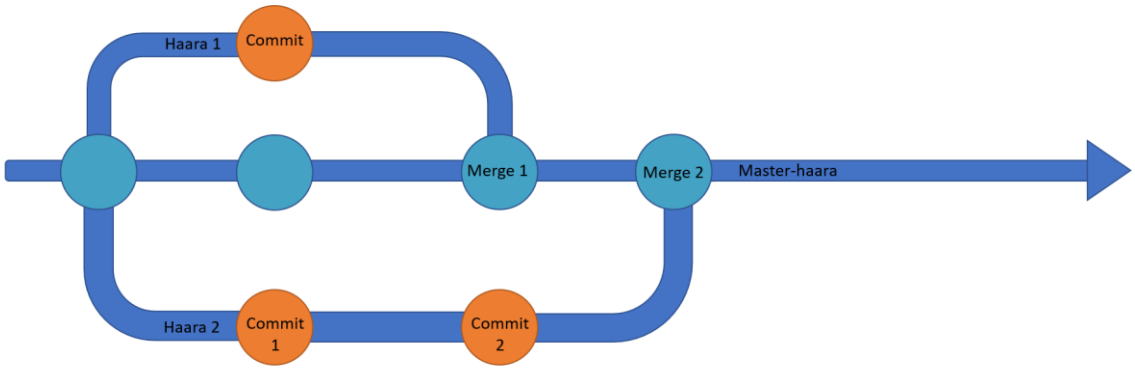


Kuva 13. Hajautetun versionhallinnan toiminta (mukaillen Malmsten, 2010)

Kuvassa 13 nähdään hajautetun järjestelmän rakenne. Ulkoinen varasto sijaitsee serverillä, johon kaikki kehittäjät saavat yhteyden tarvittaessa. Paikallinen varasto sijaitsee kehittäjän omalla tietokoneella ja on kopio ulkoisesta varastosta. Indeksi on kokoelma tiedostoja, joissa on tilastotietoja ja joiden sisältö tallennetaan objekteina. Työpuu/työkopio sisältää paikalliset muutokset, joita ei olla vielä kommitoitu sekä tiedostot, jotka sijaitsevat kyseisessä versiohaarassa. (Helsingin Yliopiston tietojenkäsittelytieteen osasto, 2019)

4.3 Versiohaarat

Versionhaarat (Branch) on versionhallintajärjestelmässä kehityksen aktiivinen osa. Jokaisessa versionhallinnassa on vähintään Master- eli päähaara. Päähaaran lisäksi versionhallintajärjestelmässä voi olla jokaista ominaisuutta, tehtävää tai korjausta varten omia haarojaan. (Helsingin Yliopisto) Rakenne voidaan parhaiten kuvata puuna, jonka runko on päähaara ja erilliset versiohaarat on rungosta lähteviä oksia. Kuvassa 14 on kuvattu versiohaaroja ja näiden suhdetta päähaaraan. Tämän lisäksi jokaisesta haarasta voi lähteä edelleen useita haaroja, jotka myöhemmin yhdistetään.



Kuva 14. Versiohaarojen suhde päähaaraan

Walrad ja Strom (2002) mainitsee, että uusi versiohaara toimii jatkuvan kehityksen lähtötilanteena. Tämä tarkoittaa sitä, että jokainen kehityshaara lähtee eristetystä tilanteesta. Eristetyllä tilanteella tarkoitetaan tilannetta, jossa kehittäminen on kehittäjäkohtaista eikä tehdyt muutokset näy päähaarassa ennen yhdistämistä ja julkaisua. Phillips, Silliton ja Walker (2011) mainitsee, että versiohaaroja on useampaa eri tyyppiä. Julkaisuhaaroja käytetään tuotteen tiettyjen versioiden ylläpitoon, prototyyppihaaroja käytetään eristettyyn kokeiluun, jonka lopputulos ei välttämättä päädy tuotantoon. Ominaisuushaaroja käytetään tuotettavan kokonaisuuden jakamista osiin. Myös bugien korjaukseen ja yhdistämiseen käytetään heidän mukaansa omia haaroja.

Versiohaarojen käyttämistä pidetään hyvänä toimintatapana, mutta Phillipsin ja muiden mukaan versiohaarojen käyttö aiheuttaa myös varjopuolia kehitykseen. Suurimpana varjopuolena he mainitsevat ”Integraatio helvetin”, jolla tarkoitetaan muutosten yhdistämisestä aiheutuvaa työtä. Tutkimuksen mukaan 9 % vastaajista eivät käytä projekteissaan laisinkaan haaroja juuri kyseisen ongelman takia. (Phillips et al. 2011) Ongelmana heidän tutkimuksensa mukaan on yhdistämisessä tapahtuvat konfliktit. Konfliktit syntyvät, kun tiedostoa kehitetään versiohaarassa, jonka aikana kehitettävän tiedoston alkuperäinen tiedosto muuttuu päähaarassa. Näin tapahtuu silloin, kun toinen kehittäjä on kehittänyt omassa haarassaan samaa tiedostoa ja päähaaran versio päivitetään uudempaan.

Tässä työssä ei keskitytä versionhallinnan haarojen tyyppeihin, mutta lopputuloksissa otetaan huomioon yhdistämisen ongelma ja versiohaarojen tärkeys.

5 Tutkimusmenetelmät ja aineistot

Tämä kappale aloittaa työn empiirisen osuuden. Osuus on koostettu empiirisen tutkimusmenetelmän esityksestä, haastatteluaineiston esittelystä, haastatteluaineiston yhteenvedosta ja tästä johdetusta kehityskohteiden määrittelystä. Kehityskohteisiin yritetään löytää ratkaisuja ja kehittäviä toimenpiteitä empiriaosuuden lopputuloskappaleessa. Lopputuloskappaleessa esitetään myös teoriaosuuden pohjalta rakennettua yleistä versionhallinnan rakennetta ja toimintatapaa.

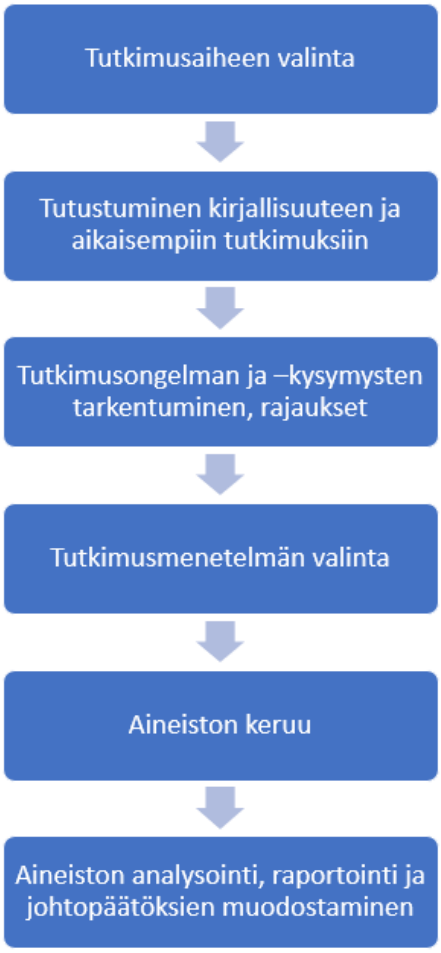
Tässä tutkimuksen empiirisessä osassa pyritään lähestymään työn tutkimuskysymyksiä puolistrukturoidulla haastattelulla, joka on toteutettu laadullisena eli kvalitatiivisena tutkimuksena. Laadullisessa tutkimuksessa aineisto pyritään keräämään monia näkökulmia antavista lähteistä, joiden avulla saadaan luotua kattava kuvaus tutkittavasta ilmiöstä. (Aira, 2005) Tutkimukseen on valittu tämä lähestymistapa, sillä puolistrukturoitu haastattelu antaa tilaa haastattelijalle johtaa ja tarkentaa kysymyksiä, mikäli haastateltava ei kysymystä täysin ymmärrä. Puolistrukturoitu haastattelu antaa selkeän linjan haastattelun rakenteesta ja linjan mitä kysymyksiä täytyy käsitellä. Useamman haastattelun tapaustutkimuksella on pyritty tilanteeseen, jossa päästäisiin tutkimaan yleisemmällä tasolla tutkimuksen rajauksien mukaisten tietovarastoympäristöjen ETL kehitystä sekä siihen liittyvää versionhallintaa ja ajankäyttöä.

Haastatteluissa mukana on ollut kohdeorganisaation kokeneita tietovarastoinnin asiantuntijoita, joten käsitteet ja määrittelyt ovat sekä haastattelijalla että haastateltavilla samat. Työssä pidetyt haastattelut on nauhoitettu sekä litteroitu.

5.1 Tutkimusprosessi

Tämä empiirinen tutkimusosuus on toteutettu monitapaustutkimuksena, joka on tapaustutkimuksen yksi tutkimusmuoto. Tutkimuksen tavoitteena oli tutkia versionhallinnan toimintatapoja suurissa yrityksissä, joissa ETL-kehityksen apuna käytetään jotain markkinoilla olevaa työkalua aiheuttaen omat haasteensa ja ristiriitaisuuden versionhallinnan käyttöön ja sen

hyödyllisyyteen. Tämän koko tutkimuksen tutkimusprosessista on tehty havainnoiva esitys Kuva 15.



Kuva 15. Tutkimusprosessi (Grönfors, 2011 & Kiviniemi, 2018)

Vaikka kuvassa esitetäänkin prosessi suoraviivaisena, on Raine Vallin julkaisemassa kirjassa Kiviniemen (2018) kirjoittaman tekstin mukaan prosessi kehittyvä. Kun tutkija on itse tutkimuksessa aineistonkeruun väline, kehittyy tutkimusprosessin edetessä hänen tietoisuutensa ja näkemykset tutkittavaan asiaan. Tutkimusprosessi alkaa tutkimusaiheen valinnalla, joka on tehty tämän työn määrittelyvaiheessa. Tässä vaiheessa tärkeintä on löytää tutkimusaiheeksi sellainen aihe, joka kiinnostaa tutkijaa. (Grönfors, 2011) Prosessi jatkuu aiheen kirjallisuuteen ja aikaisemmin tehtyjen tutkimuksiin tutustumisella, jossa tutkijalle syntyy kokonaiskuvaa tutkittavasta aiheesta. Samalla tutkija kykenee tarkentamaan tutkimusongelmaansa, määrittelemään tutkimuskysymykset ja rajaukset tutkittavan aiheen ympärille.

Kun tarkempi määrittely on saatu tehtyä tutkimusaiheen ympärille, pystyy tutkija myös päättämään parhaimmakseen näkevän tutkimusmenetelmän, jolla suorittaa aineiston keruun. Viimeisenä vaiheena tutkija analysoi ja raportoi aineistonsa luettavaan muotoon ja muodostaa johtopäätöksensä keräämänsä aineiston pohjalta tukeutuen aikaisemmin tutkimaansa kirjallisuuteen. Tässä vaiheessa on kuitenkin muistettava, että kaikkea aineistoa, jota on kerännyt ei kannata sisällyttää tutkimusraporttiin. (Kiviniemi, 2018) Kiviniemi myös lisää, että on tärkeää sisällyttää tutkimusraporttiin se minkälaista kehitystä tutkijassa, aineistonkeruumenetelmässä tai tutkittavassa ilmiössä tapahtuu, jotta ulkopuoliset lukijat pystyvät arvioimaan vaihtelevan tutkimusprosessin hallintaa.

5.2 Tutkimusmenetelmän esittely

Kuten tämän työn aikaisemmissa vaiheissa on mainittu, tutkimus on toteutettu usean tapauksen tutkimuksena eli monitapaustutkimuksena, joka on yksi tapaustutkimuksen muodoista. Monitapaustutkimus valikoitui työn tavoitteen takia, sillä monitapaustutkimuksella voidaan perehtyä tarkemmin useampaan tapaukseen ja näiden avulla luoda yleistys, jolla tarkastella laajempaa tutkimusaiheena olevaa aihetta.

Päivärinta (2019) mainitsee pro gradu työssään, että monitapaustutkimus on tapaustutkimuksen tyyppi, jossa tutkitaan useampaa kohdetta rinnakkain osana suurempaa kokonaisuutta tai ilmiötä. Hän myös lisää, että monitapaustutkimuksessa analysoidaan usean saman aihepiiriin liittyvää kohdetta, joita voidaan nimetä osatapauksiksi. Nämä osatapaukset ovat tässä tapauksessa organisaatioita ja tarkemmin organisaation sisällä tietovarastoinnin tiimejä ja toimintatapoja.

Tässä tutkimuksessa aihepiirinä oli ETL kehityksen ominaispiirteet, organisaation sisäinen toimintatapa, työkalun aiheuttamat haasteet versionhallintaan sekä itse ETL-kehityksen versionhallinta. Tutkimukseen rajauksiksi esitettiin työn alussa kolme eri rajausta. Ensimmäinen rajaus oli, että yritys käyttää ETL kehitykseen jotain ETL työkalua. Yleisesti graafisten käyttöliittymien työkalut muodostavat tiedostot xml tai vastaavassa tiedostotyypissä ja täten yksinkertainen koodin yhdistäminen versioiden välillä on joko haastavaa tai mahdotonta. Toiseksi rajaukseksi esitettiin datan hyödyntämisen, eli ETL kokonaisuuksien suuri määrä, jotta versionhallinnalla on konkreettista hyötyä ja merkitystä. Viimeisenä rajauksena

tutkimuskohteille esitettiin se, että kehityksessä käytetään useampaa ympäristöä ja näillä on oma konkreettinen funktio kehityksessä ja ylläpidossa. Viimeisen rajauksen tosin pitäisi kehittyneessä tietovarastoinnin ja ETL kehityksen ympäristössä olla itsestäänselvyys, kuten teoriakappaleissa on perusteltu.

Tämän monitapauksen osatapauksina toimivat Enfo Oyj:n rajauksien mukaisten asiakasyrityksien tietovarastointitiimi, joskin haastatteluissa on haastateltu kyseisissä asiakasyrityksissä työskenteleviä Senior konsultteja, jolla on useamman vuoden työkokemus ja vankka tieto haastateltavasta aihepiiristä. Tutkimukseen valitut asiakasyritykset on valittu aikaisemmin mainittujen rajauksien mukaisesti, mutta myös siten, että niistä saadaan erilaista tietoa ja vaihtelevuutta asiasisältöön. Monitapaustutkimuksen haastattelut on nauhoitettu ja ne on litteroitu luettavaan muotoon, jotta yleisen analyysin tekeminen on helpompaa. Kysymykset on pidetty kaikilla haastateltavilla samoina, joskin haastattelija on ohjannut ja tarkentanut kysymyksiä, mikäli tähän on ollut tarvetta.

Päivärinta (2019) mainitsee, että tapaustutkimuksissa yleistäminen on yksi rajoitteista, sillä tapausotanta on liian pieni verrattuna reaalimaailmaan, joskin aineistoa pystytään analysoimaan teoreettisesti osana ilmiötä ja näin ollen omaa yleistettävyyttä jollakin tasolla. Robert Stake (2015, s.23) mainitsee omassa kirjassaan, että monitapaustutkimuksen hyödyt häviävät, mikäli osatapauksia valitaan vähemmän kuin neljä tai enemmän kuin 10. Samalla vaikka tapauksista ei voida tehdä yleistystä, on jokainen osatapaus kuitenkin mahdollisuus oppia tutkittavasta aiheesta. (Stake 2015, s.24)

Tässä tutkimuksessa kerättiin tietoa puolistrukturoiduilla haastatteluilla, joiden kesto vaihteli noin 45 minuutista reiluun tuntiin. Puolistrukturoidussa haastattelussa kysymykset ovat valmiina ja avoimia, joten haastateltava vastaa kysymyksiin omin sanoin ja oman tietämyksensä perusteella. Tästä syystä haastateltavaksi valikoitui viisi alalla ollutta senior konsulttia, joiden tietotaidolla ja asiakasymmärryksellä saatiin luotua hyvä kuvaus asiakasprojektien ETL ympäristön tämänhetkisestä tilanteesta.

Puolistrukturoiduissa haastatteluissa haastattelun runko on suuntaa antava mutta haastattelun kulku on strukturoitua vapaampi. Täten haastattelija voi keskustelun edetessä tarkentaa kysymyksiä, kysyä järjestyksestä eroten kysymyksiä tai kokonaan haastattelurungossa olemattomia kysymyksiä, joilla voidaan kerätä lisätietoa. Haastattelukysymysten määrä on rajattu tiiviiksi, jotta haastattelijalla on mahdollisuus kysyä tarkentavia kysymyksiä ja tässä

tapauksessa esitellä työhön liittyvän ohjelman konseptitodistusta, jolla pystyttäisiin korjaamaan joitain kyseisen asiakasprojektin ongelmakohtia.

Kuten Päivärinta (2019) on omassa pro gradu työssään haastatterlurakenteen esitellyt, on se tehty samalla tavalla myös tässä tutkimuksessa (Liite 1). Ensimmäisenä haastattelu aloitetaan haastateltavan profiiliin suunnatuilla kysymyksillä. Seuraavaksi on haastattelun varsinainen sisältö, eli sisältökysymykset, joilla pyritään keräämään tarvittavat tiedot laajempaa tutkimusaiheena olevaa aihetta tarkastellakseen. Viimeisenä osana haastattelurungossa on lopetuskysymykset, joiden tavoitteena on varmistaa, ettei haastateltavan mielestä jotain merkityksellistä ole jäänyt käsittelemättä ja viedä haastattelu loppuun.

5.3 Tutkimusaineisto

Tämän tutkimuksen aineisto koostuu viidestä haastattelusta, jotka on pidetty tammikuussa 2022. Haastatteluita on pidetty samoina päivinä, mutta ne on nauhoitettu ja litteroitu myöhäisempää analysointia ja raportointia varten. Haastatteluihin valittiin Enfo Oyj:n omia tietovarastoasiantuntijoita, joilla on työkokemusta ja tarvittavaa tietotaitoa. Näin haastateltavat pystyivät vertaamaan nykytilannetta muihin vastaaviin, vaikka keskittyminen tässä tutkimuksessa koskikin juuri tällä hetkellä meneillään olevaa projektia ja sen ominaispiirteitä ETL-kehityksen versionhallinnassa.

Haastatteluihin valitut henkilöt työskentelevät asiakasprojekteissa, jotka täyttävät työn alussa määritellyt tutkimusrajaukset

1. ETL kehitykseen käytetään jotain markkinoilla olevaa ETL-kehitystyökalua
2. Yrityksessä käytetään dataa suuremmissa määrin, joten ETL kokonaisuuksia on suurempi määrä
3. Organisaatiossa käytetään useampaa ympäristöä kehitykseen ja ylläpitoon

Haastatteluihin valittiin useasta organisaatiosta haastateltavia, sillä oli tärkeää saada monipuolisia vastauksia, jotka eivät kaikki noudata samaa kaavaa kehitysprosessissaan, eivät käytä samaa työkalua eivätkä samanlaista versionhallintamenetelmää tai työkalua. Näin ollen, vaikka aineiston keruussa onkin vain viisi haastateltavaa, saadaan tutkittava käsiteltyä

laajemmalla skaalalla. Taulukossa 8 nähdään haastatteluiden päivämäärät, haastateltavien tietovarastoasiantuntijoiden nimet sekä tehtävänimikkeet, joita tämä tapaustutkimus koskee.

Taulukko 8. Haastatellut asiantuntijat

Päivämäärä	Nimi	Tehtävänimike	Haastattelun kesto
11.1.2022	Pasi Äikäs	Senior konsultti	63 min
11.1.2022	Olli Rahunen	Senior konsultti	51 min
13.1.2022	Petri Alho	Senior konsultti	48 min
13.1.2022	Petri Lindgren	Senior konsultti	55 min
14.1.2022	Seppo Laukasto	Senior konsultti	50 min

Tutkimuksessa saatiin kerättyä tietoa ja ominaispiirteitä ETL-kehityksen ympäristöistä, tähän liittyvästä henkilöstömäärästä ja siitä millä tavalla versionhallintaa ja sen toimintatapoja voisi kehittää, jotta työskentely olisi sujuvampaa, virheriskittömämpää ja nopeampaa. Tutkimuksen aineiston raportointi ja analysointi esitellään myöhemmässä vaiheessa. Työssä ei määritellä mistä organisaatiosta mitkäkin vastaukset on saatu, vaan kaikista vastauksista on luotu yhtenäinen vastauskokonaisuus. Näin ollen tutkimukseen on saatu yhteenveto osatapauksien vastauksista ja yleistettyä nämä vastaukset osaksi kokonaisuutta, kuten monitapaustutkimuksessa on tarkoitus.

Tutkimusta jälkeenpäin arvioidessa olisi voitu yhdeksi rajoitukseksi asettaa myös ETL-kehittäjien määrän. Täten olisi voitu tehdä johtopäätöksiä ja yleistystä myös eri kokoisissa kehittämisympäristöissä ja ottaa huomioon tämän vaikutusta kokonaistilanteeseen. Kehittäjien määrä on yksi merkittävimmistä tekijöistä, kun huomioitavana osa-alueena on versionhallinnan sujuvuuden sekä merkityksen tärkeys ja kehitysprosessi. Nyt tutkimuksessa on suurta hajontaa kokoluokassa ja näin ollen saadaan tehtyä yksi johtopäätös sekä pienistä että suurista ETL-kehitysympäristöistä muttei näiden eroista.

6 Haastattelutulokset ja kehityskohteiden määrittely

Tässä kappaleessa käsitellään puolistrukturoidun haastattelun tuloksia, mutta kuten aikaisemmin mainittiin, tämä raportointi on tehty niin ettei vastauksista voi tunnistaa yritystä tai haastateltavaa vaan kaikkien haastatteluiden vastaukset on yhdistetty kysymyskohtaisesti yhdeksi yleiseksi vastaukseksi. Näin ollen tässä kappaleessa käydään läpi kysymyksien vastaukset ja näistä tehdyt johtopäätökset ja vastausten erilaisuus. Vaikka sisältökysymyksiä ei välttämättä kysytty samassa järjestyksessä, käydään ne tässä raportointivaiheessa samassa järjestyksessä, kun ne on liitteeseen 1 merkitty.

6.1 Haastattelutuloksien esittely

1. Kuinka monta ihmistä toimii projektissa ETL kehittäjänä tavalla tai toisella?

Haastatteluissa ilmeni, että tutkimuksessa olleet asiakasprojektit työllistivät jollain tapaa ETL kehityksessä mukana olevia työntekijöitä neljästä (4) seitsemääntoista (17). Voidaan siis suoraan sanoa, että ETL ympäristöihin ja työkaluihin liittyvät vaatimukset voivat olla hyvin erilaisia. Juuri tästä syystä yhtenä tutkimusrajauksena olisi voinut olla kehittäjien määrä. Tältä osin kuitenkin pystytään tekemään havaintoja yleisesti ETL ympäristöistä näiden ennalta asetettujen rajausten mukaisesti ja ottaa huomioon pienempien projektien kohdalla potentiaali skaalautua suuremmaksi toiminnan kehittyessä. Tämä henkilöluku pitää sisällään itse kehittäjät ja mallintajan, joskin mallintaja ei aina tee varsinaista ETL-kehitystä, eikä näin ollen tarvitse versionhallintaa apunaan. Joissain tapauksissa mallintaja toimii myös kehittäjänä, joten tämäkin toimenkuva täytyy ottaa huomioon tutkimusta tehdessä.

2. Millainen ETL-kehityksen prosessi on tällä hetkellä alusta loppuun?
    a. Kokonaan uuden kokonaisuuden rakentamisessa?

Kaikissa osatapauksissa kokonaan uuden kokonaisuuden rakentaminen tehdään yleisesti samalla tavalla kuten kuvattu kuvassa 8. Uuden kokonaisuuden kehittämisen prosessi alkaa siitä, kun liiketoiminnalta tulee tarve tietylle taululle tai datalle. Tästä tarpeesta tehdään

ennakkomäärittely ja mallintaja mallintaa taulun, avaimet ja siihen liittyvät dimensiot. Mallinnuksen jälkeen kehittäjä ottaa työn tehtäväkseen ja kehittää kokonaisuutta kehitykseen tarkoitetussa ympäristössä. Joissain tapauksissa ympäristöjä ei ole kolmea, vaan kehitykseen käytettävä ympäristö on samalla myös testausympäristö. Kehittämisen jälkeen riippuen kyseisen ympäristön datasta, sitä joko testataan samassa ympäristössä tai siirretään yhden tason ylemmäs ympäristöön, jossa tuotantodataa tai tuotantodataan verrattavaa dataa on saatavilla. Haastatteluista ilmeni, kuten myös teoriaosiossa mainittu on kehitys ja testaus iteratiivinen prosessi, eli kokonaisuutta testataan ja kehitetään limittäin ja vuoron perään. Yleisesti viimeinen vaihe on tuotantoonsiirto ja viimeinen testaus tässäkin ympäristössä. Osassa haastatteluista mainittiin, että testaaminen tuotannossa on vaarallista, koska kokonaisuuden yksi prosessi saattaa jäädä jumiin ja kuluttaa tuotantopalvelimen suorituskapasiteetin aiheuttaen jumeja ja muita ongelmia.

b. Olemassa olevan ylläpidossa ja/tai muokkauksessa?

Tässä haastattelukysymyksessä muodostui kolme ryhmää tutkimuksessa mukana olleitten osatapausten mukaan siitä miten paljon suhteessa ylläpitoa ja uuden tuottamista tehdään. Ensimmäisessä ryhmässä mainittiin, että jo olemassa olevan kokonaisuuden muokkausta harvemmin ilmenee. Toisessa ryhmässä ylläpitoa tehdään yhtä paljon kuin uutta toteutusta ja kolmannessa ryhmässä ylläpitotyötä on pääosa kaikesta kehittämisestä.

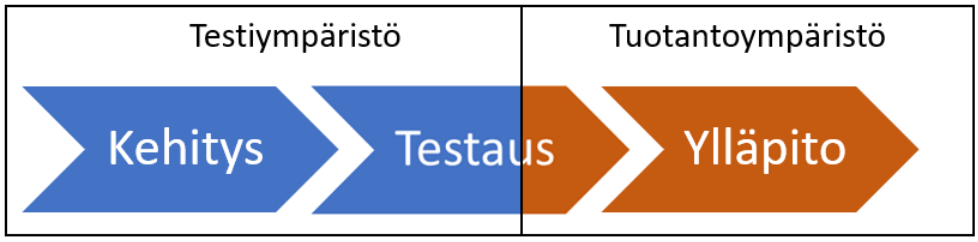
Haastatteluvastauksista saatiin tehtyä johtopäätös, että yleisesti toimintaprosessi alkaa liiketoiminnan toiveesta tai korjaustarpeesta. Kun tämä toive ja korjaustarve on määritelty, saatetaan korjaus tehdä suoraan tuotantoympäristöön, mikäli vaikeusaste sen sallii. Jos kyseessä on ollut uuden kentän lisäämistä tai laskentalogiikan muuttamista, tämä viedään mallimuutoksena tietomalliin vasta korjauksen jälkeen. Korjaustarpeesta syntynyt mallimuutos jätetään kokonaan tietomallista pois, mikäli laskentalogiikka ei ole muuttunut.

3. Kuinka montaa ympäristöä käytetään? (esim. Kehitys, Tuotanto, Testi jne.)

Ympäristöjen lukumäärää ja näiden käyttötarkoitusta tutkiessa tultiin tällä otannalla siihen tulokseen, että yleisempi toimintatapa on ylläpitää kahta eri ympäristöä joissa toisessa kehitetään ja testataan ja toinen ympäristö on liitetty tuotantodataan. Yhdellä näistä osatutkimuskohteista oli käytössä kuitenkin kolme eri ympäristöä, joiden käyttötarkoitukset olivat teoriaosuudessakin mainitut kehitys, testaus ja tuotanto.

a. Mikä on näiden jokaisen merkitys kehitysprosessissa ja kokonaisuudessa?

Kahden ympäristön kokonaisuuksia voidaan kuvata tämän tutkimuksen osatapauksien osalta kuvan 16 merkitsemällä tavalla.



Kuva 16. Tutkimuskohteen kahden ympäristön funktio

Testiympäristöä käytetään uuden ETL-latauksen kehitykseen ja siellä on myös tarkoitus testata pääasiassa latauksen toiminta, ennen kuin se viedään tuotantoympäristöön. Kuten sanottu, joissain tapauksissa testiympäristön data ei vastaa tuotantodataa esimerkiksi dimensiotauluihin liitettäessä, joten testausta täytyy tältä osin jatkaa myös tuotantoympäristössä. Tähän sisältyy tietenkin riski, että lataus poistaa jotain tuotantoympäristölle tärkeää dataa, mutta näin on joka tapauksessa päädytty tekemään esimerkiksi laillisten velvoitteiden vuoksi.

b. Miten versioiden siirtäminen ympäristöstä toiseen tapahtuu ja miten paljon tähän kuluu aikaa?

Versioiden siirtäminen ympäristöstä toiseen tulee raskaammaksi siinä vaiheessa, kun yhtä tiedostoa joudutaan siirtelemään ympäristöstä toiseen useammin kuin kerran tai jos kerralla joudutaan siirtämään useampi tiedosto useasta eri kansiopolusta. Tämä on myös riippuvainen käytetystä ETL työkalusta, sillä joissain markkinoilla olevista työkaluista siirtely on tehty helpoksi työkalun omilla toiminnallisuuksilla. Toisissa työkaluissa tätä ominaisuutta ei ole, vaan kaikki siirtämisen vaiheet joudutaan tekemään käsin kehittäjän toimesta. Tämä kehittäjän manuaalisesti tekemä työ on kehityksen kannalta turhaa, joskin ympäristöjen toiminnan kannalta tärkeää. Siirtely aiheuttaa myös riskinsä, esimerkiksi jos tuotantoympäristössä kehitetään latausta, mutta sitä ei siirretä normaalisti kehityksessä käytettävään testiympäristöön, voi seuraava kehittäjä kehittää edellistä, testiympäristössä olevaa versiota ja ylikirjoittaa tuotannon version uudestaan testiympäristössä kehitetyllä versiolla.

4. Mitä ETL-työkalua käytetään ja onko kyseisellä työkalulla jotain ominaispiirteitä?

Tutkimuksessa huomattiin, että joissain osatutkimuskohteista käytetään samaa työkalua kuin toisessa, mutta pääosin ominaispiirteenä voidaan sanoa olevan graafisella käyttöliittymällä muodostettujen latausten tiedostomuoto. Tiedostomuodot ovat joko XML (Extensible Markup Language) tai sitä muistuttavaa tiedostotyyppiä. Joissain työkaluissa on myös sisäänrakennettu oma versionhallinta, joka näyttää eri versioiden numeroinnit ja muokkauspäivämäärät.

   a. Pystytäänkö samaa tiedostoa/kokonaisuutta kehittämään samaan aikaan ja muutokset yhdistämään, vai pitääkö tiedosto ns. lukita yhdelle kehittäjälle?

Haastateltavissa organisaatioissa käytettävät työkalut eroavat toisistaan tämän kysymyksen kohdalla siten, että joissain työkaluissa työstettävä tiedosto lukitaan yhdelle kehittäjälle kerrallaan ja toisissa yhtä tiedostoa pystyy kehittämään useampi kehittäjä samanaikaisesti. Tällaisissa työkaluissa, joissa kehitystä pystytään tekemään rinnakkain, aiheutuu riski, että uusin tallennus ylikirjoittaa vanhemman esimerkiksi muutama minuuttia sitten tehdyn tallennuksen. Tästä syystä oma kehittäjäkohtainen kehitysympäristö on tärkeä työkaluissa, joissa lukitusominaisuutta ei ole. Toisaalta työkalut, jotka lukitsevat kehitettävän kohteen täytyy tuo lukitus poistaa itse, jotta muut kehittäjät pystyvät jatkamaan kehitystä. Tässä haastattelussa ilmenneiden seikkojen vuoksi pystytään kuitenkin vetämään johtopäätös, että kehityksellisestä näkökulmasta omaa kehitysympäristöä ei välttämättä tarvita, mikäli lukitseminen on työkalun oma ominaisuus.

   b. Millä tiedostotyypillä ympäristöstä tuotu tiedosto tallennetaan?

Kaikkien tämän monitapaustutkimuksessa mukana olleiden osatapauksien ETL työkalut tuottavat tiedostotyypin, joka muistuttaa XML tiedostoa. Koska tiedosto on XML tietotyyppiä vastaava, se koostuu elementeistä ja tämän attribuutista. Tiedostossa on siis jonkinlainen selkeä rakenne, jonka sisälle varsinainen tuotettu sisältö asetetaan. Kuten mainittua, tämä aiheuttaa oman haasteensa versionhallintaan, jota ei pystytä käyttämään täydellä potentiaalilla mahdollisten konfliktitilanteiden ilmentyessä. Konfliktitilanne syntyy, kun samaa tiedostoa on kehitetty rinnakkain ja versionhallintajärjestelmä huomaa eron, joka pitäisi ratkaista. Normaalissa tekstimuotoisessa tiedostossa on mahdollista yhdistää nämä muutokset toisiinsa, jotta saadaan molemmat muutokset mukaan viimeiseen versioon. XML

tietotyypissä tämä ei kuitenkaan ole mahdollista, joten versioiden yhdistäminen ilman siihen kehitettyä työkalua ei onnistu.

5. Mikäli asiakasprojektissanne hyödynnetään versionhallintaa:

    a. Miten kuvailisit nykyistä versionhallinnan toimintakokonaisuutta ja toimintamallia?

Vastaajat olivat omien asiakasprojektien versionhallintajärjestelmistä sitä mieltä, että ne toimivat omalla tavallaan juuri siihen käyttötarkoitukseen, kun ne on asiakasprojektissa määritelty. Useassa vastauksessa kuitenkin ilmenee, että versionhallinnan käytössä on puutteita, se on aikaa vievää ja joissain vastauksissa korostui myös monimutkaisuus komentojen käytössä. Asiakasprojektissa, jossa ei käytetty mitään erillistä versionhallintaa, mukaan olisi haluttu erillinen versioiden välinen vertailu, jossa tiedostojen sisältöä voisi vertailla koko tiedoston näkökulmasta korostaen esimerkiksi eri väreillä poistettuja ja lisättyjä kohtia.

    b. Mitä versionhallintajärjestelmää käytetään?

Tutkimuksessa haastattelijoiden vastaukset vaihtelivat tässäkin työssä mainittujen kahden versionhallintajärjestelmän, Gitin ja SVN:n eli Subversionin välillä. Yhdessä asiakasprojektissa ei käytetty laisinkaan erillistä versionhallintajärjestelmää. Tässä asiakasprojektissa luotettiin työkalun omaan versiointiin ja tuotantoympäristöstä haettuun varmuuskopioon, joka tehdään manuaalisesti tietyn aikavälin syklillä.

    c. Missä funktiossa versionhallintaa pidetään?

Erillistä versionhallintaa pidetään pääasiassa varmuuskopiona, jota ylläpidetään päivittäin usean kehittäjän toimesta. Versionhallinnan kautta pystytään tarkastelemaan tehtyjä muutoksia historian aikana, mutta sitä ei suurilta osin käytetä kehityksen tukena, muissa kuin juuri mainituissa asioissa. Yhdessä asiakasprojektissa kehittäminen alkaa versionhallinnan käytöllä, kun tehdään uusi versiohaara ja vedetään paikalliseen varastoon uusimmat muutokset jatkaen tästä itse kehitykseen.

    d. Jos käytössä on versiohaaroja, mikä on näiden nimeämislogiikka?

Niissä projekteissa, joissa ei ole muutamaa kehittäjää enemmän, ei käytetä versiohaaroja sillä niistä ei nähdä olevan suurta hyötyä itse kehitystyön kannalta. Kyseisessä tilanteessa

kaikki mukana kehityksessä olevat henkilöt tietävät mitä kokonaisuutta kukakin työstää, joten ristiriitoja ei synny kehityksen edetessä. Samalla näissä projekteissa työkalun sisältämä lukitus poistaa edelleen näitä riskejä.

Projektit, joissa versiohaaroja käytetään, nimetään ne yleisesti tehtävienhallintaohjelmiston tehtäväkohtaisilla nimityksillä. Näin ollen jälkeenpäin versionhallintajärjestelmästä nähdään mitä muutoksia on tehty tiettyä tehtävää kohden. Samaten versionhallinnasta nähdään kaikki tehtävät, jotka ovat liittyneet kyseisen kehitettävän kohteen historiaan.

e. Mikä on versionhallinnassa pisin manuaalinen prosessi?

Niissä yrityksissä, joissa ei käytetä kehitykseen versiohaaroja, pisin manuaalinen prosessi on tiedostojen siirtäminen ympäristöstä toiseen. Tämä tehdään manuaalisesti ja on mahdollista, että tiedostoja pitää siirtää useammasta eri kansiosta. Täten kehittäjän täytyy itse pitää kirjalla muutettuja tiedostoja, jottei tuotantoon viennin yhteydessä kokonaisuudesta puutu jokin toiminnan kannalta merkittävä osa.

Yrityksissä, joissa käytettiin versiohaaroja, pisin manuaalinen prosessi on käytännössä koko versionhallinnan käyttö. Tämä pitää sisällään uuden versiohaaran luomisen sekä paikalliseen että ulkoiseen varastoon, ympäristöjen välisen siirtelyn sekä ulkoiseen varastoon viemisen ja haarojen yhdistelyn.

6. Jos asiakasprojektissa ei hyödynnetä erillistä versionhallintaa, onko näin tehty tarkoituksella? Miksi? Onko versionhallinnan hyödyntämistä mietitty tai suunniteltu asiakasprojektissa?

Tällaisia osatapauksia, jossa ei käytetty erillistä versionhallintaa oli vain yksi, joten tällä otannalla ei voida tehdä johtopäätöksiä. Todennäköisiä syitä on mahdollinen lisäkustannus ja näkemys tarpeettomuudesta.

7. Minkälaisia haittoja tai hyötyjä näet asiakasyritykselle, mikäli versionhallinnan tukena olisi työkalu?

Kaikki osatutkimuskohteet, joissa käytettiin versionhallintaa, olisi sitä mieltä, että työkalusta versionhallinnasta olisi hyötyjä. Työkalulla saataisiin kynnys versionhallinnan jatkuvalle

käytölle madallettua helppokäyttöisyyden ansiosta. Tähän lisättiin, että työkalun toiminta on aina samanlaista kaikille kehittäjille, joten virhealttius madaltuisi ja versionhallintaa käytettäisiin kaikkien kehittäjien toimesta samalla tavalla. Versionhallintaa versiohaarojen kanssa käyttäville organisaatioille työkalu toisi helpotusta nykyiseen toimintamalliin, jossa kehittäjä katsoo listaa tai ohjeita ja suorittaa komennot manuaalisesti. Listaa seuraamalla virheiden ilmentyessä merkitys kasvaa, kun kehittäjä palaa takaisin listaan pohtimaan mikä vaihe on jäänyt välistä.

Isoin ja yleistettävin hyöty työkalusta olisi kuitenkin siinä, että manuaalisen versioiden siirtelyn saisi automatisoitua, eikä tähän menisi kehittäjiltä turhaa aikaa. Haastattelujen aikana työkaluun saatiin kuitenkin kehitysehdotusta, että työkalussa olisi hyvä olla monivalintavalikko kaikille muuttuneille tiedostoille.

8. Millaiselle asiakkaalle erillinen versionhallinta olisi järkevä ratkaisu?

Kysyttäessä millaiselle asiakkaalle erillinen versionhallinta olisi järkevä ratkaisu vastaukset keskittyivät teknologiaan, jota käytetään sekä siihen miten suuri kokonaisuus on hallittavana kohteena sekä ihmis- että tiedostomäärällisesti. Omassa painoarvossaan täytyy kuitenkin pitää sitä, miten helposti muutoksia pystytään seuraamaan historiaan ja toimintavarmuus myös esimerkiksi tuotantopalvelimen korruptoituessa. Skaalautuvuutta ja kasvupotentiaalia täytyisi myös miettiä, joskin versionhallinnan käyttöönoton voi tehdä myös projektitiimin jo kasvaessa.

9. Onko asiakasprojektin versionhallintajärjestelmässä jotain puutteita?

Suurimmaksi puutteeksi kaikissa haastatteluissa pidettiin automatisoinnin puutetta ja sitä, että kehittäjän täytyy itse pitää mielessä mitä tiedostoja on muutettu tuotantoon siirron vaiheessa. Tähän myös saatiin vastaukseksi kehitettävän kohteen tarkastuksen puuttumista. Mikäli tällainen lisävaihe otettaisiin käyttöön versionhallinnan pakottamana, siirtyisi tietotaito ja tekninen osaaminen myös muille kehittäjille.

6.2 Kehityskohteet ja huomiot

Tässä välikappaleessa määritellään tutkimuksessa selvinneet ja käsiteltäväksi otettavat kehityskohteet. Kehityskohteita määritellessä otetaan huomioon niiden merkittävyys versionhallinnan ja koko prosessin käytössä sekä huomiot, joita haastattelututkimuksen aikana saatiin esille. Huomioista suurin ja yleispätevin oli se, että mikäli haastatteluissa selvinneiden nykytilanteiden toimintaa voidaan pitää hyvänä, ei erillistä versionhallintaa asiakasprojekteissa tarvita välttämättömyytenä. Näin ollen, mikäli kehittäjämäärä on vähäinen ja/tai kehittäjät tietävät mitä tiedostoja tai kokonaisuuksia toiset kehittävät, ei erillistä versionhallintaa tarvita. Tässä huomiossa on kuitenkin pidettävä mielessä toiminnan kasvamisen mahdollisuus. Jos kehittäjiä yhtäkkiä tarvitaankin enemmän, voi koko kehitystiimi kohdata ongelmia versionhallinnan puutteen takia. Toinen tutkimuksessa ilmennyt havainto oli, että osassa projekteissa käytetään pelkästään ETL työkalun omaa versionhallintaominaisuutta. Jatkotutkimuksia olisi hyvä tehdä siitä, riittääkö työkalun oma versionhallinnan ominaisuus koko tiimin versionhallintaan. Jatkotutkimuksessa pitäisi huomioida myös sitä, kuinka monta versiota työkalu muodostaa yhdellä kehityskerralla, jossa iteratiivisesti kehitetään kokonaisuutta testauksen ja kehityksen suhteen. Koska jokaisella tallennuskerralla ja iterointikierroksella syntyy uusia versioita, voi muiden kehittäjien olla vaikea palata viimeiseen toimivaan versioon, mikäli uusi versio ei toimi halutulla tavalla enää myöhemmin. Nämä molemmat huomiot liittyvät erillisen versionhallinnan tarpeellisuuteen ja voidaankin sanoa, että ensimmäistä huomiota voidaan pitää validina myös työn lopputuloksia miettiessä. Työkalun omaan versionhallintaa ei jatkossa huomioida, sillä kaikissa markkinoilla olevissa työkaluissa ei tällaista toiminnallisuutta ole, eikä se kaikissa tapauksissa ole riittävä koko kehitystiimin tarpeisiin.

Kehityskohteita määritellessä tutkimuksessa ilmeni neljä seikkaa, mitkä asiakasprojekteissa kaipaavat kehitystä tai uudelleen suunnittelua. Ensimmäinen näistä on se, että niissä asiakasprojekteissa, joissa luotetaan työkalun omaan versiointiin, ei pystytä näkemään koko tiedostoa, kun katsotaan eri versioiden välisiä eroja. Tämä ongelma korjaantuu sillä, että otetaan käyttöön esimerkiksi joko SVN tai GIT versionhallintajärjestelmä, joissa tämänkaltainen ominaisuus on. Toinen kehityskohde oli se, että versionhallinnan sekä versioiden siirtäminen ympäristöjen välillä on täysin manuaalista työtä, eikä tähän ole työkalua. Kehityskohteena tämä sopii täydellisesti tähän työhön, sillä työn esimäärittelyssä päätettiin, että

tavoitteena on luoda työkalu tukemaan versionhallinnan käyttöä ja poistamaan turhaa manuaalista työtä. Tämä tehdään ohjelmoimalla konseptitodistus työkalusta, jolla versionhallinnan saa sidottua kehitykseen tiiviimmin. Samalla työkalulla saadaan poistettua eroavaisuuksia versionhallinnan käytössä. Kolmas kehityskohde oli tutkimuksessa selvinnyt seikka, että kehittäjän täytyy itse muistaa, mitä tiedostoja on muokattu yhden ETL putken toimimista varten. Tämä aiheuttaa lisää manuaalista työtä, sillä kehittäjä toimii samalla omana muistikirjanaan joko kirjoittamalla muutetut tiedostot ylös tai muistellessaan näitä. Työn tavoitteena olevan työkalun pitäisi siis olla kykenevä havainnoimaan muutetut tiedostot. Neljäntenä kehityskohteena voidaan pitää kehitysprosessia, jossa kehitettävän tiedoston tai koodin tarkastusta toisen kehittäjän toimesta ei pidetä suuressa merkityksessä. Mikäli kehittäjät tarkastelisivat toisten kehittäjien kehittämää kokonaisuutta, tulisi ylläpidosta huomattavasti helpompaa ja yleinen tietotaidon jakaminen osaksi kokonaisprosessia. Tältä osin tämä kehityskohde liittyy tutkimusaiheeseen, mikäli se saadaan liitettyä osaksi versionhallintaa tai kehitettävää työkalua.

Viimeinen, jopa merkittävä huomio tämän työn aihepiiriin on se, että kaikissa tutkimuksessa olleissa osatapauksissa käytettiin ETL työkalua, joiden tiedostot tallennetaan joko XML-tiedostotyypissä tai vastaavassa. Tämä tarkoittaa sitä, että yhtä tiedostoa ei voi kehittää kuin yksi kehittäjä, eikä näin ollen versionhallinnassa päästä käyttämään versionhallinnan täyttä potentiaalia kehityshaarojen yhdistämisessä. Samaten myöskään itse kehityksessä ei päästä samaan tehokkuuteen, kuin kehittäessä jollain ohjelmointikielellä, jossa tiedosto ei koostu tietystä rakenteesta. Tämä rajoite aiheuttaa sen, että versionhallintajärjestelmällä ei ole suurta merkitystä, koska samanaikaista kehittämistä ei päästä tekemään. Näin ollen, sekä SVN, että GIT ovat työn aihepiiriin ja koko ETL kehitykseen nähden yhtä valideja valintoja versionhallintajärjestelmiksi.

7 Lopputulokset

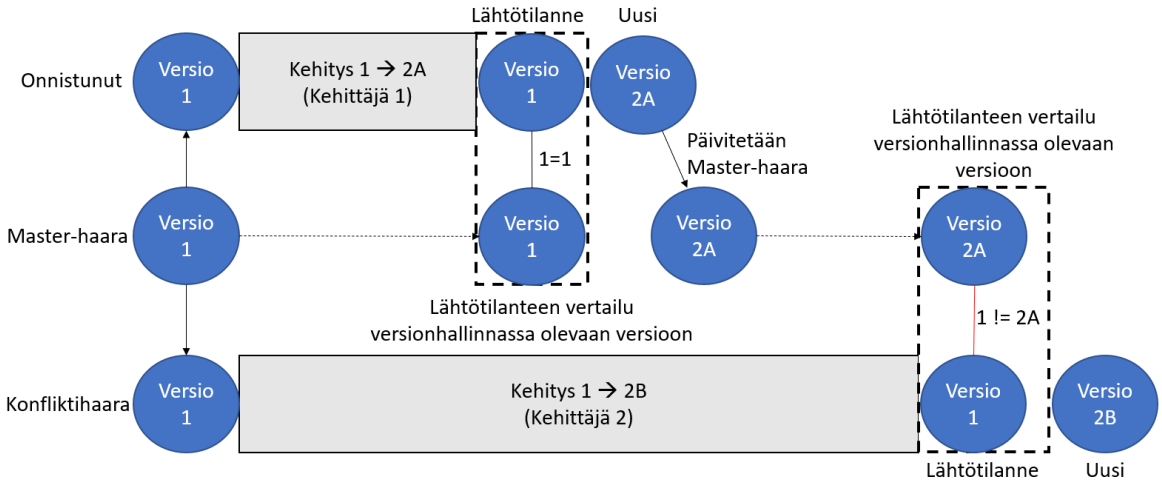
Tässä kappaleessa käydään läpi tutkimuksen aikana kehityskohteiksi nousseiden asioiden ratkaisuja, joita tehtiin. Ensimmäisenä esitellään yleistetty teoriaosuuteen pohjautuva versionhallintamalli, jossa on otettu huomioon myös työn empiirisen osan huomioita. Malli on tehty, jotta versionhallinnalla saadaan konkreettista hyötyä esimerkiksi versioiden välisten erojen havainnoinnilla, tarkastusprosessin käyttöönotolla sekä varmuuskopioina, jotka sisältävät kehityshistorian alusta loppuun. Toisena esitellään työn aikana yhteen asiakasprojektiin kehitetty konseptitodistus työkalusta, jolla saatiin ratkaistua GIT:in hankala ja manuaalinen käyttö käyttöliittymällä toimivalla työkalulla.

7.1 Yleinen versionhallinnan toimintamalli

Työn ensimmäisenä lopputuloksena on teoriaan ja haastatteluilla kerätyn aineiston perusteella muodostettu yleinen versionhallinnan toimintamalli. Toimintamalli tehdään siten, että sen rakenne on teoriaan pohjautuva ja jolla saadaan poistettua haastatteluissa ilmenneet ongelmat versioiden välisessä tarkastelussa ja toimintatavoissa.

Kuten aikaisemmin on esitetty, toisen versionhallintajärjestelmä sukupolven suosituin järjestelmä SVN lukitsee tiedostot, kun kehittäjä alkaa kehittämään näitä. Seuraavassa sukupolvessa, hajautetuissa järjestelmissä tätä ei tapahdu. Koska ETL työssä tiedostotyypit ovat rakenteellisia tekstitiedostoja kuten XML, ei tässä tapauksessa lukitsemisella ole merkitystä. Merkityksen ETL työssä poistaa se, että rakenteellisten tiedostotyyppien tiedostojen versioita ei pystytä yhdistämään toisiinsa, joka sinällään toimii yhtenä lukkona kehittämisessä. SVN myös lukitsee tiedostot vain tiettyyn versiohaaraan, eli mikäli usealla kehittäjällä on oma versiohaaransa, pystytään kiertämään lukitus ja päästään lukituksen kanssa samaan lopputulokseen kuin hajautettujen versionhallintajärjestelmien kanssa. Tältä osin työn lopputuloksissa voidaan vetää johtopäätös, että sekä toinen että kolmas versionhallintajärjestelmäsukupolvi ovat yhtä valideja, mikäli verkkoyhteyden toimivuutta ei oteta huomioon. Verkkoyhteyden voidaan taas olettaa olevan toiminnassa, koska työtä tehdään tavalla, joka vaatii yhteyden sekä tietovarastoon että eri ympäristöjen palvelimille.

Mikäli versionhallintajärjestelmissä käytetään versiohaaroja, voidaan törmätä ongelmaan yhdistelyssä, mikäli kaksi kehittäjää ovat kehittäneet samaa tiedostoa eri haaroissa. Tämä ilmenee yleisesti siten, että ensimmäinen kehittäjä yhdistää oman kehitetyn versiohaaransa päähaaraan, jolloin jälkimmäinen kehittäjä yhdistäessään saa ilmoituksen konfliktista. Konflikti tässä tapauksessa tarkoittaa sitä, kun kuvan 17 kehittäjä 2 yhdistäessään kehittämää haaraansa törmää tilanteeseen, jossa päähaarassa (Master-haara) oleva versio ei vastaa enää sitä versiota, josta kehittäjä on lähtenyt omaa versiotansa kehittämään.



Kuva 17. Konfliktitilanne yhdistäessä haaroja

Tämä on ongelma tehtävienhallinnassa, eikä niinkään itse versionhallinnan käytössä. Ongelma johtuu siitä, että samaa versiota kehitetään samaan aikaan, vaikka tiedetään ettei näiden yhdistäminen ole mahdollista. Siksi tässä työssä ei keskitytä tämän ongelman ratkaisemiseen, joskin voidaan todeta, että optimaalisella tehtävienhallinnalla ongelmaan ei törmätä. Versiohaarojen käytöstä syntyy siis usean kehittäjän ympäristössä mahdollisia tilanteita, jotka hidastavat kehittämistä. Silti versiohaarojen käyttö on järkevää, sillä jokainen versiohaara on oma eristetty systeemi suuremman versionhallinnan sisällä. Konfliktitilanteet myös poistavat mahdollisuuden, jossa toisen kehittäjän tallennus ylikirjoitetaan.

Versiohaarojen käytössä järkevintä on käyttää tehtävä- tai toimintokohtaista nimeämistä. Tämä tarkoittaa sitä, että versiohaara nimetään sen mukaan mitä korjaus tai lisäys koskee. Yleisesti tämä tarkoittaa tehtävienhallintajärjestelmiin luotujen tehtävien kirjain ja numerosarjaa, jotta versionhallintajärjestelmästä voidaan tutkia tiedostojen historiamuutoksia tehtäväkohtaisesti. Samaten pystytään palaamaan tilanteeseen ennen tiettyä muutosta, mikäli lopputulos ei olekaan sellainen, kun se on suunniteltu olevan. Mikäli kehittäjät eivät käytä

tällaista järjestelmää, on järkevintä nimetä versiohaarat lisättävän ominaisuuden mukaan. Versiohaarojen käytössä täytyy kuitenkin muistaa poistaa ne haarat, jotka on yhdistetty päähaaraan. Tämä siistii kokonaisuutta helppolukuisemmaksi ja mikäli haarat listataan, nähdään mitkä ominaisuudet tai tehtävät ovat kehityksessä eivätkä vielä valmistuneet. Versiohaarojen luomislogiikassa täytyisi pidättäytyä mallissa, jossa haaroja tehdään vain päähaaran lisäksi yhdelle tasolle. Näin ollen yhdistämistä ei tarvitse tehdä ylimääräisiä kertoja, vaan ainut yhdistäminen tehdään, kun valmis ja testattu kokonaisuus yhdistetään päähaaraan.

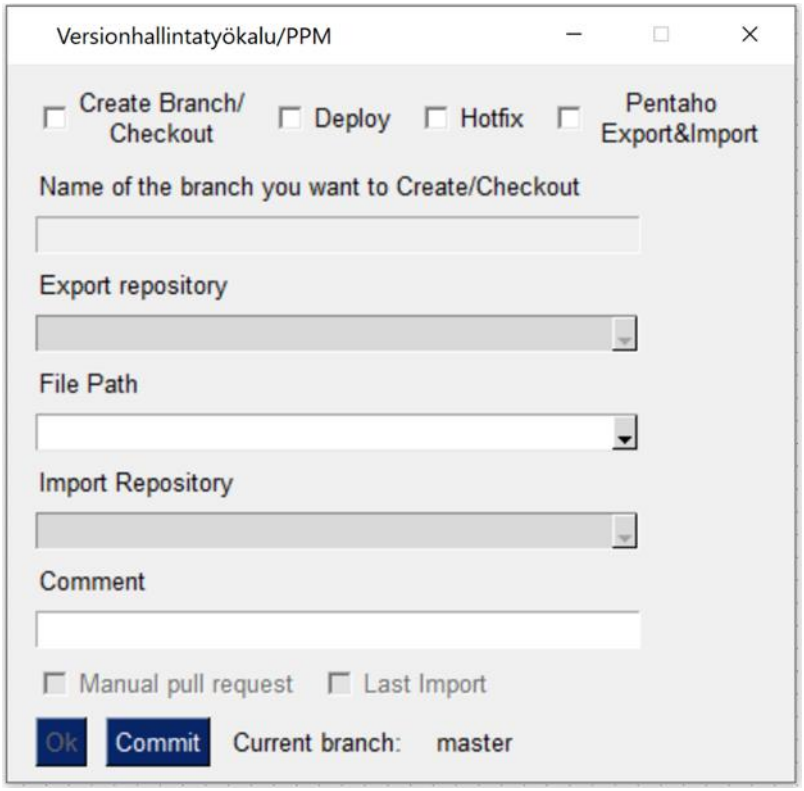
Toimintatapojen osalta suurempi muutos versionhallinnan käyttöön on commit komennon käyttö, joka tallentaa ulkoisen varaston versiohaaraan nykytilanteen kehittäjän paikallisen varaston vastaavasta haarasta. Näin ollen saadaan sisällytettyä tarkemmin tehtyjen muutosten sarja. Tämä tarkoittaa sitä, että yhdistäessä versiohaaraa päähaaraan täytyy kehittäjän käyttää komentoa ”--no-ff” eli ei pikakelausta. Kyseinen komento tarkoittaa sitä, että se sisällyttää viimeiseen yhdistämiseen kaikki erilliset kommitoinnit, eikä vain yhteenvetoa. Toimintatapoihin toinen muutos on se, että versiohaaroja yhdistäessä tehdään vetopyyntö (pull request) jolla ohjataan kokonaisuus tarkastettavaksi jollekin toiselle kehittäjälle. Vetopyynnössä kehittäjä pyytää toista kehittäjää tarkastamaan ja yhdistämään versiohaaran päähaaraan, jolloin muutokset siirtyvät päähaaraa vetämällä myös muille kehittäjille. Näin osaamista saadaan levitettyä myös tarkastajalle ja samalla poistettua koodiin jääneitä virheitä tai suorituskykyongelmia.

Tähän yleiseen versionhallintamalliin voidaan sisällyttää myös alla kuvailtu konseptitodistus työkalusta, joka tehtiin yhdelle tutkimuksessa mukana olleista asiakasprojekteista. Työkalulla on tarkoitus poistaa manuaalisten vaiheiden kuormaa sekä poistaa virhealttiutta. Samaten työkalun avulla on tarkoitus madaltaa kynnystä versionhallinnan käyttöönotossa ja päivittäisessä käytössä.

7.2 Proof-of-Concept työkalu

Kuten aikaisemmin työn tavoitteissa mainittiin, toisena lopputuloksena on aikaisemmin esitelty yksinkertainen versionhallinnan toimintamalli, jota voidaan hyödyntää useampaan ympäristöön. Toisena lopputuloksena tässä työssä saatiin suunniteltua ja toteutettua ohjelma, jolla yhden asiakkaan versionhallinnan ongelmakohtia saatiin vähenemään ja toimintaa

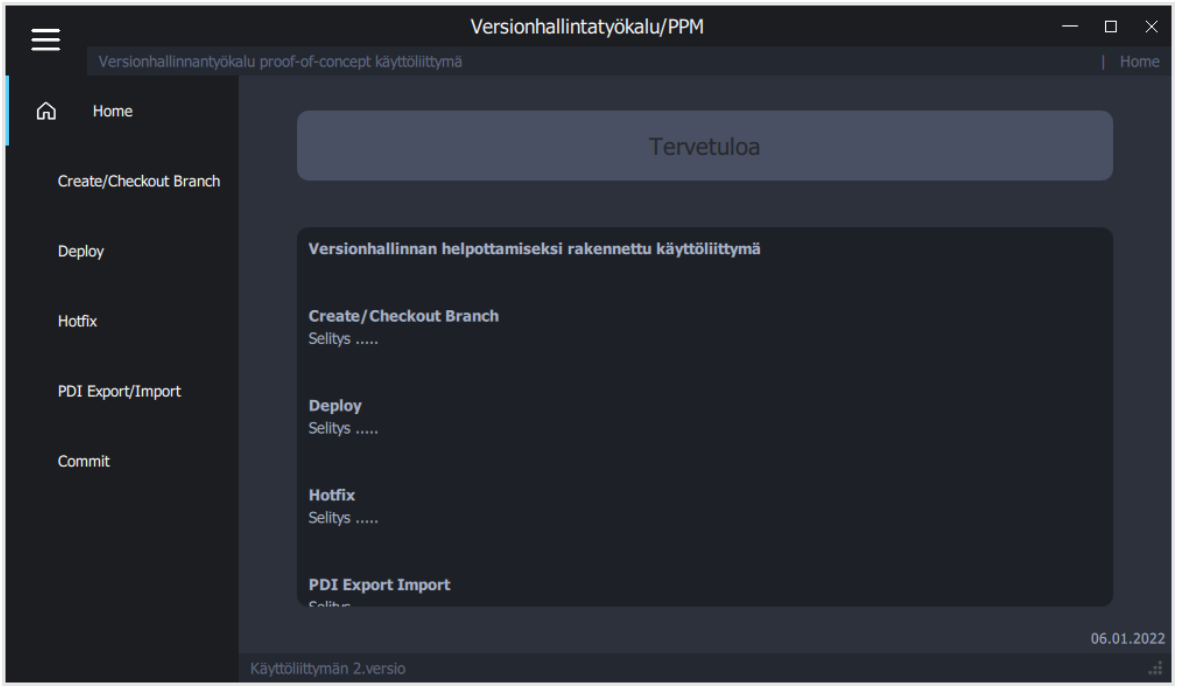
helpotettua ja yksinkertaistettua. Tämä lopputulos on työn alussa mainittu konseptitodistus, jonka kehittämisellä pyrittiin vähentämään kehittäjien manuaalista työtä sekä virhealttiutta versionhallinnan kanssa toimiessa. Ohjelmaan sisällytettiin myös versioiden siirtely (Export&Import ja Deploy) toiminnot. Syy tähän on se, että nämä siirrot toteutettiin ennen ohjelman tekemistä manuaalisesti ETL-työkalua hyödyntäen. Toimintojen selitykset esitetään myöhemmin ja lisäksi kerrotaan, kuinka nämä toteutettiin ennen. Tarkempaan ohjelman sisältämän koodin esittelyyn ei tässä työssä keskitytä, sillä ohjelma on toteutettu esittämään vain toiminnallisuuksien mahdolliset resurssi- ja ongelmatilanteiden aiheuttamat aikasäästöt. Näitä säästöjä on verrattu aikaisempaan manuaaliseen toimintatapaan, jossa käytettiin apuna joko Git Bashia tai käyttöliittymän avulla toimivaa työkalua kuten Sourcetree.



Kuva 18. Versionhallintatyökalun 1. käyttöliittymä

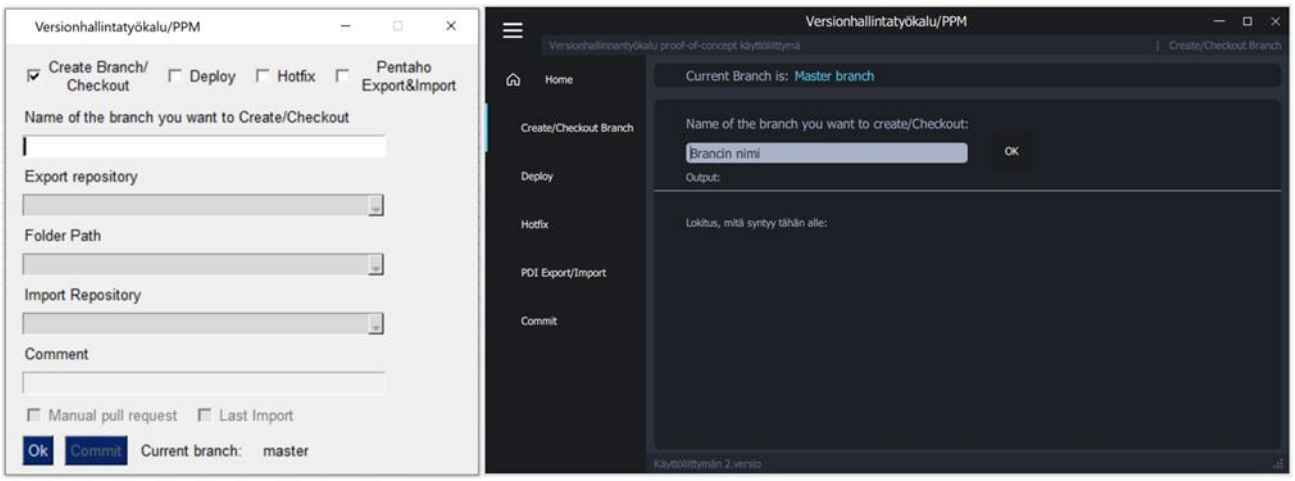
Kuvassa 18 esitetään versionhallintaan tehdyn ohjelman käyttöliittymä, joka on tehty hyvin yksinkertaiseksi tässä tuotantoympäristössä olevassa versiossa. Tämä tehtiin siitä syystä, että yksinkertainen käyttöliittymä on helpompi käyttää ja se oli nopeampi tehdä, joten itse toiminnallisuuksien ohjelmointiin saatiin käytettyä tarpeellinen aika. Ohjelmaan on sisällytetty virheidenhallintaa esimerkiksi kirjoitusvirheiden, tyhjien kenttien tai versionhallinnan päähaarassa olemisen varalle. Virheen kohdattuaan ohjelma luo pop-up ikkunan, jossa

kuvaillaan kohdattu ongelma ja ratkaisu. Pop-up ikkuna ilmestyy myös, kun tietty toiminto-kokonaisuus on suoritettu. Onnistuneen toiminnon pop-up ikkunassa mainitaan mitä tehtiin, millä parametreillä ja että se on valmistunut.



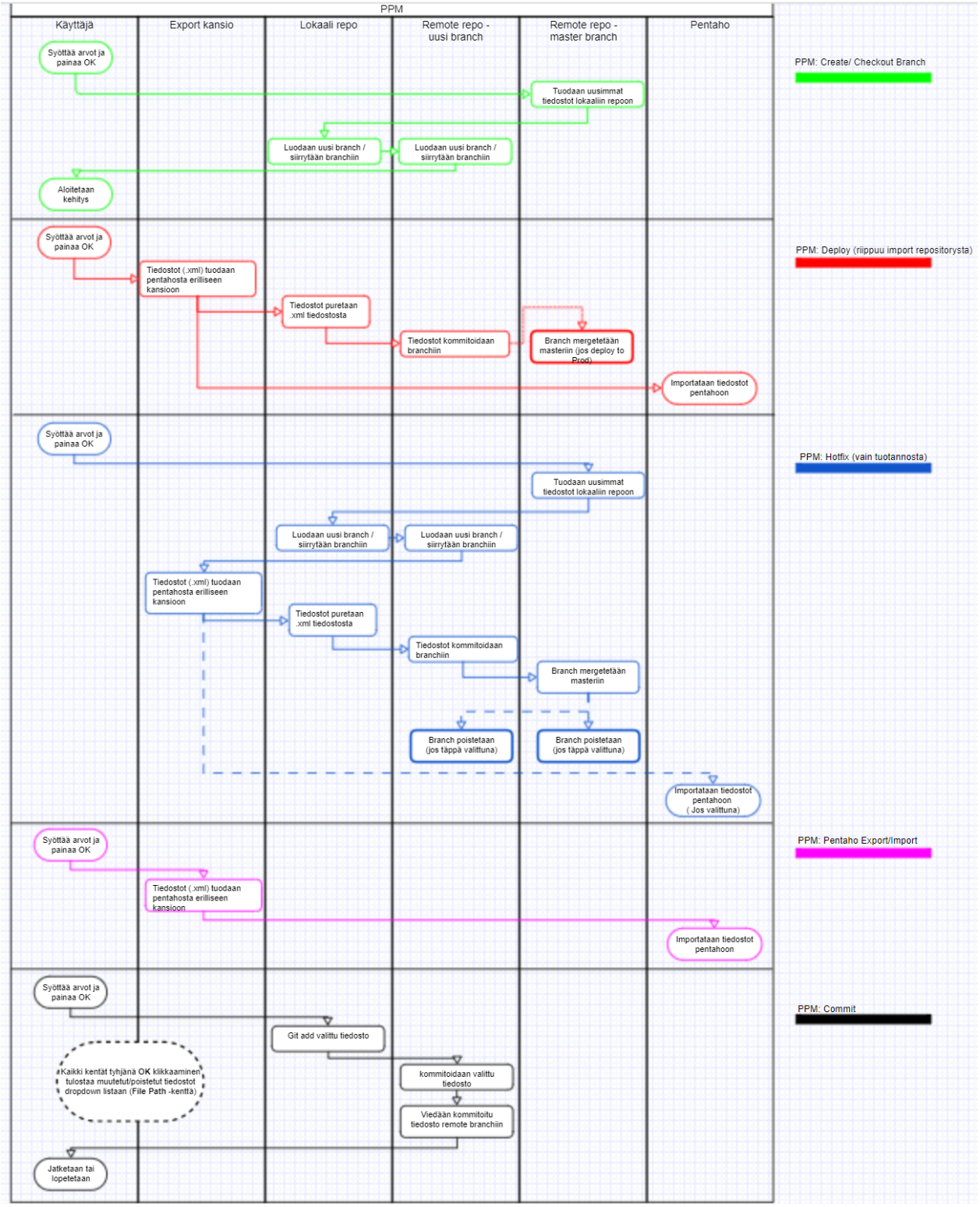
Kuva 19 Versionhallintatyökalun 2. käyttöliittymä

Kuvasta 19 nähdään kehitysiteraatiossa toinen ohjelman käyttöliittymä, jolla saadaan tehtyä ohjelmasta modernimman näköinen sekä kaikki tarvittava ohjeteksti yhdelle kotisivulle. Tässä toisessa käyttöliittymän kehityksessä on käytetty Qt Groupin luomalla raahaa ja pudota metodilla toimivalla työkalulla Qt Designer. Tämä valikoitui ohjelmaksi, sillä toisen ja kehittyneemmän käyttöliittymän tarkoituksena oli esittää seuraava askel käyttöliittymän kehityksessä. Näin ollen varsinaisen ohjelmointikielen harjoittelu olisi ollut työn aihepiirin ja työmäärän osalta liian aikaa vievä osio. Työssä käytettiin Qt Designerin ilmaista lisenssiä, joten se ei sinällään kaupalliseen tarkoitukseen sovellu, mutta ajaa asiansa uuden ja edistyneemmän käyttöliittymän esityksessä.



Kuva 20. Eri käyttöliittymien vertailu ja erot

Kuvassa 20 on kuvattu näiden kahden käyttöliittymän erot vielä valitessa Create/Checkout Branch toiminto. Ensimmäisessä versiossa valinta tehdään ylärivillä olevan valintaruutujen avulla, joita voi olla valittuna vain 0 tai 1 samaan aikaan. Tässä toiminnossa käyttäjä ei voi syöttää kuin uuden tai olemassa olevan versiohaaran nimen, koska tämä on ainut tieto mitä tarvitaan. Samalla tavalla myös uudessa versiossa ainut syötettävä tieto on versiohaaran nimi, joskin käyttöliittymään on lisätty tulostuskenttä johon ohjelma tulostaa mitä on tehty ja onko se onnistunut. Uudessa käyttöliittymässä on moderniin tapaan lisätty sivupaneeli, jonka avulla saadaan navigoitua eri toimintojen välillä, jokaisella välilehdellä on näkyvillä nykyinen versiohaara ja esillä vain kentät, joita kyseiseen toimintoon tarvitaan. Sivupaneelin saa myös suljettua vasemman ylänurkan hampurilaislogon avulla sekä antaa visuaalisen palautteen sinisellä merkillä missä toiminnossa ollaan sen lisäksi, että oikean ylänurkan paneelissa lukee valittu toiminto/välilehti.



Kuva 21 Versionhallintaohjelman toiminnot uimarata kaaviossa.

Ohjelman käyttöliittymästä huomataan helposti neljä toimintoa, mutta se sisältää myös viidennen eli Commit toiminnon. Toiminto on tärkeämpi kehityksessä, jonka kokonaiskesto on pitkä ja välitallennukset tarpeellisia. Jokainen eri toiminto on esitetty kuvassa omalla värillään ja katkoviivoilla esitetään vaihetta, joka tehdään, mikäli näin on valittu.

Vihreällä kuvassa 21 on esitetty uuden versiohaaran luominen tai olemassa olevaan siirtyminen (checkout). Tässä toiminnossa ohjelma katsoo, löytyykö syötetyn versiohaaran nimistä versiohaaraa ja mikäli sellaista ei löydy, se luodaan sekä kehittäjän omaan että palvelimelta löytyvään varastoon. Tämän jälkeen paikallisen varaston versiohaara asetetaan seuraamaan palvelimelta löytyvää vastaavaa. Mikäli versiohaara on olemassa, tehdään normaali siirtyminen ja ladataan uusimmat muutokset.

Punaisella on merkitty Deploy toimintoa. Toiminnon ennakkovaatimuksena on se, että on luotu jo kehitettävää kokonaisuutta varten oma versiohaaransa ja tehty tarvittavat muutokset joko testaamista tai tuotantoon vientiä varten. Toiminnossa työkalu tuo tietyn kansion haluamasta ympäristöstä (testi tai kehitys) ja vie sen testaamista varten testiympäristöön tai tuotantoon vientiä varten tuotantoympäristöön. Samalla kokonaisuus viedään palvelimelta löytyvään versionhallintaan. Mikäli kyseessä on tuotantoon vienti, se myös asetetaan tarkastettavaksi vetopyynnöllä tai yhdistetään suoraan päähaaraan, mikäli erillistä tarkastusta ei tarvita.

Sinisellä on merkattu niin sanottu Hotfix eli korjaus, joka on tehty suoraan tuotantoon. Nämä korjaukset ovat yleisesti pieniä korjauksia, joissa korjattava kohde on helposti havaittavissa ja korjattavissa. Hotfix- toiminto yhdistää aikaisemmin mainitut versiohaaran luomisen ja tuotantoon viennin. Hotfix toiminnossa tehdään uusi versiohaara, jonka avulla viedään versionhallintaan uusi tuotannosta löytyvä kokonaisuus ja asetetaan se review tilaan tai yhdistetään päähaaraan. Useasti tämä korjattu kohta on niin yksinkertainen, ettei erillistä vetopyyntöä ja tarkistusta tarvita. Vaaleanpunaisella kuvaan on merkitty Export/Import, joka ei liity versionhallintaan mutta on lisätty työkaluun kehittäjien avuksi. Tässä ETL kokonaisuus siirretään ympäristöstä toiseen, jotta manuaalisen työn määrä vähenisi myös versionhallintaan kuulumattomassa työssä.

Mustalla on merkitty aikaisemmin mainittu Commit. Kuten kerrottu, tätä toiminnallisuutta ei välttämättä tarvita, mikäli kehittäjä kehittää kokonaisuuden itse alusta loppuun nopeasti, sillä commit toiminto vie palvelimen varaston versiohaaraan paikalliseen varastoon tehdyt uusimmat muutokset.

7.3 Työkalulla saavutetut hyödyt ja jatkokehitys

Työkalulla tavoiteltiin versionhallinnan helpompaa nitomista ETL kehityksen yhteyteen. Tässä kappaleessa esitetään, miten mikäkin ohjelman vaihtoehdoista vastaa tavoitteisiin ja miten työkalua voisi kehittää paremmaksi.

Työkalun suurin ongelma on tällä hetkellä se, että se on toteutettu teknologiariippuvaiseksi. Ohjelma käyttää toimiakseen yhden tutkimuksissa olleiden organisaatioiden ETL työkalua, koska tässä työkalussa ETL kokonaisuuden siirtäminen vaatii työkalun omaa toimintoa paikallisen varaston synkronointiin. Ongelma on helposti ratkaistavissa muokkaamalla ohjelmakoodia niissä tapauksissa teknologiariippumattomiksi, joissa ei vaadita työkalun omia toimintoja. Tämä voitaisiin tehdä käyttämällä esimerkiksi SSH tiedonsiirtoprotokollaa, jolla tiedostot siirrettäisiin ympäristöistä toisiin. Lisäksi haastatteluiden aikana selvisi, että ohjelmaan olisi monikäyttöisyyden vuoksi lisättävä monivalintavaihtoehto, joka listaisi kaikki muuttuneet tiedostokansiot. Täten työkalulla voitaisiin siirtää useampia kansionsisältöjä samalla kertaa, eikä kehittäjän tarvitsisi itse pitää kirjalla mitä on muutettu ja mitä pitää viedä ympäristöstä toiseen. Tämä on tarpeellinen lisäys työkaluun, vaikka se ei olekaan välttämätön kaikissa ETL-kehitysympäristöissä.

Työkalun Create/Checkout branch suorittaa tarkistuksen, onko syötetyn nimistä versiohaaraa olemassa ja toimii tämän tuloksen pohjalta. Jos versiohaaraa ei ole, sellainen tehdään ja paikallinen versiohaara asetetaan seuraamaan ulkoisessa varastossa olevaa samaa haaraa. Näin ollen paikallista haaraa päivittäessä muutokset haetaan juuri tuosta ulkoisen varaston samasta haarasta, eikä koko versionhallinnasta. Tällä toiminnolla saadaan siis useampi manuaalisesti suoritettava komento yhteen syöttökenttään ja Ok-painikkeeseen.

Työkalun Deploy on työkalun monimutkaisin toiminto. Manuaalisesti tehtynä on useampi toimintatapa, mutta yleisesti ETL-kehityksen prosessin mukaan versionhallintaa käytettäessä prosessi aloitetaan luomalla uusi versiohaara kehitystä varten ja päivittämällä

paikallinen varasto. Tämä toiminto on äsken esitelty Create Branch. Tämän jälkeen kehitys tapahtuu normaalisti siihen asti, kunnes on testaamisen vuoro. Manuaalisesti tehtynä pitäisi kokonaisuus tuoda kehitysympäristöstä kehittäjän omalle tietokoneelle. Tämän jälkeen kehittäjän tulisi erikseen kirjautua testiympäristöön ja viedä kokonaisuus omalta tietokoneelta testiympäristön kansioihin. Ohjeiden mukaisesti tässä vaiheessa olisi hyvä viedä testiversio versionhallintaan, joka lisää manuaalisten komentojen määrää. Samat manuaaliset komennot tehdään myös tuotantoon viennin yhteydessä. Erona tuotantoon viennin vaiheessa on lisättävä pakollinen versionhallintaan siirto ja joko versiohaaran yhdistäminen päähaaraan tai vetopyynnön luominen. Työkalussa yksi ympäristöjen välinen siirto tapahtuu yhdellä klikkauksella ja kolmella syötettävällä kentällä. Nämä kolme kenttää on: mistä siirretään, mitä siirretään ja minne siirretään. Työkaluun on lisätty myös toiminnallisuus, jossa testiin viennissä ei yhdistetä versiohaaraa. Tuotantoon siirrossa haara yhdistetään ja lisäksi on mahdollista valita, halutaanko tarkastusprosessi mukaan vai hyväksytäänkö muutokset suoraan päähaaraan. Tämä on ylivoimaisesti eniten aikaa säästävä toiminnallisuus, sekä virheiden määrän pitäisi pysyä nollissa, mikäli kehittäjät eivät ole kehittäneet samaa haaraa aiheuttaen konfliktia versioiden välille.

Hotfix toimii tuotannosta-versionhallintaan siirrossa eli käyttää deploy-toiminnallisuuden kanssa samoja toimintoja. Hotfix siirtää tiedostot tuotantoympäristöstä paikalliseen kehitysympäristöön, josta tiedostot viedään versionhallintaan. Manuaalisesti tehtynä tämä vaihe sisältää sekä versionhallinnan manuaaliset komennot että kokonaisuuden tuomisen paikalliseen varastoon.

Kun työkalun näkymä on tyhjä, voi kehittäjä painaa ”commit” -painiketta, jolloin työkalun auetessa syntyvään erilliseen ikkunaan ilmestyy kaikki muuttuneet tiedostot. Näin ollen, kehitysvaiheessa olevan tiedoston välitallennus kyseiseen versiohaaraan on tehty nopeammaksi ja helpommaksi ja saatu mukaan kehitysprosessiin. Samaten tällä ominaisuudella saadaan täytettyä työkalun kehityskohteissa määritellystä ominaisuudesta, jossa työkalun täytyisi pystyä seuraamaan muuttuneita tiedostoja. Commit -toimintoa käyttäessä tulee kuitenkin käyttää versionhallinnan eri versiohaaroja, sillä toiminnallisuus toimii sekä SVN:n että GIT:n tukemilla komennoilla ”status”, jossa versioita verrataan versiohaaraa ja sen alkuperäisiä tiedostoja vertaamalla.

Yhteenvetona voidaan sanoa, että työkalulla on päästy ennalta määriteltyihin tavoitteisiin, mutta koska kyseessä on konseptitodistus, kaipaa se ennen yleisempää käyttöönottoa vielä uusia ominaisuuksia sekä teknologiariippumattomia toimintatapoja.

8 Johtopäätökset

Digitalisaation ja datan kasvavan hyödyntämisen takia ETL kehityksen merkitys tulee kasvamaan yritysten toiminnassa. ETL prosessi on tärkeimmässä osassa, kun mietitään raakadatan muuntamista luotettavaksi lähteeksi päätöksen teon tueksi tai koneoppimismallien hyödynnettäväksi. Tällä hetkellä ETL kehitykseen käytetään tietovarastointiprojekteissa noin 70–80 % resursseista, joten kehityksen ja lopputulosten kannalta merkityksettömien manuaalisten työosuuksien poistaminen on ajankohtaista. Suuremmassa ETL kehitysorganisaatiossa versionhallinta luo sujuvuutta, auttaa kehityksen tukena ja luo erillisen varmuuskopion, jota päivitetään päivittäin. Näin ollen versionhallinnan käsittely ETL kehityksen resurssisäästöissä valikoitui tutkimuksen aiheeksi ja tutkimusaiheessa pysyttiin.

Tutkimuksen aikana yllättävää oli, ettei ETL kehityksen ominaispiirteitä olla otettu huomioon versionhallintaan liittyvissä tutkimuksissa. Jatkotutkimuksia varten tulisi ottaa huomioon se, että tässä tutkimuksessa käsiteltiin henkilömäärältään hyvin eri kokoisia kehitystiimejä. Mielenkiintoista olisi käsitellä eri kokoisten tiimien ominaispiirteitä ja parhaita toimintatapoja sekä kehitysprosessissa että versionhallinnan käytössä. Tutkimuksen loppuun on kuitenkin sanottava, että mikäli rakenteellisen tiedostotyypin yhdistely olisi mahdollista voisi koko versionhallinnan hoitaa automatisoinnilla esimerkiksi Jenkins -ohjelmiston avulla.

Tutkimuksen tavoitteena oli selvittää, kuinka saadaan säästettyä ETL kehityksessä toimivien kehittäjien aikaa versionhallinnan manuaalisista askelista itse kehitykseen sekä millainen versionhallintamalli toimisi yrityksissä, joissa versionhallinta kaipaa kehitystä tai sitä ei käytetä laisinkaan. Aihepiiriin liittyen ei löytynyt aikaisempia tutkimuksia juuri ETL kehityksen ja versionhallinnan väliltä. Tästä syystä työssä on sovellettu tutkimuksia, joissa ETL kehitys on korvattu ohjelmistokehityksellä. Näihin tutkimuksiin on otettu huomioon ETL kehityksen ominaispiirteet, jotta voidaan tehdä järkeviä johtopäätöksiä.

Käydään läpi nämä tutkimuskysymykset ja käsitellään työn aikana luodut lopputulokset sekä näiden suhde johdannossa määriteltyihin tavoitteisiin.

Tutkimuskysymys 1. Minkälainen versionhallintamalli sopii parhaiten ETL kehitysympäristöihin, joissa on useampi kehittäjä?

Versionhallinnan ja ETL kehityksen välistä tutkimusaineistoa ei löytynyt, joten määrittely tehtiin täysin teoriaosuuden ja empiriaosuuden haastatteluiden avulla. Mallissa otettiin huomioon ETL kehityksen ja työkalujen luomat ominaispiirteet ja haasteet. Koska versionhallintajärjestelmällä ei ole päivittäisessä käytössä tehtyjen oletuksien mukaan suurta merkitystä, voidaan sekä toisen että kolmannen sukupolven järjestelmiä käyttää. Työn lopputuloksista määriteltiin myös, että versiohaarojen käyttäminen olisi suotavaa, sillä näin ollen koko versionhaara toimii omana varmuuskopionaan. Täten versionhallinnan päähaaraa ei saada sotkettua, vaikka palaaminen tiettyyn kohtaan onkin mahdollista. Palautus kuitenkin loisi lisätyötä, kun tarkoitus on päästä erillisistä aikaa vievistä askelista eroon. Viimeinen yleisen versionhallinnan mallin ero, joka eroaa suurelta osin ohjelmistotuotannon versionhallinnasta, on se, että versiohaaroja tehdään vain yhteen tasoon. Näin ETL kehityksen suurin haaste, eli yhdistäminen saadaan mahdollisimman pieneen osaan.

Tutkimuskysymys 2. Kuinka saadaan kehittäjien aikaa siirrettyä manuaalisista vaiheista itse kehittämiseen?

Tällä tutkimuskysymyksellä pyrittiin juuri kaksiosaiseen lopputulokseen, ensimmäisen ollessa yleinen versionhallinnan malli. Tämän kysymyksen avulla oli tarkoitus luoda automaatiota niihin vaiheisiin, joissa kehittäjät käyttävät eniten aikaa tai joissa kehittäjät tekevät eniten virheitä. Työssä kehitettiin konseptitodistus työkalusta, jolla graafisen käyttöliittymän avulla saadaan suoritettua monen komennon ketjut yksinkertaisesti ja aina samaa logiikkaa noudattaen. Näin ollen voidaan sanoa, että tutkimuskysymykseen on vastattu. Työkalun ollessa käytössä ajansäästöllinen vaikutus on merkittävä, sillä virheiden aiheuttamiin ongelmiin ei tarvita toista kehittäjää avustamaan ratkaisussa. Samaten manuaaliset vaiheet saadaan suoritettua automaattisesti, eikä ne vaadi huomiota tai ajankäyttöä samalla tavalla kuin ilman työkalua.

Koska työn aikana luotiin virheitä vähentävä yksinkertainen työkalu, saavutettiin myös tämän tutkimuskysymyksen toisena, epäsuorana vaikutuksena se, että versionhallintaa on nyt helpompi käyttää. Näin ollen kaikki kehittäjät pystyvät käyttämään versionhallintaa samalla

tavalla kuin muut ja käyttöönoton askel on huomattavasti pienempi kuin komentorivin avulla käytettävä ja monia komentoja vaativa versionhallinnan kokonaisuudessa.

Lähteet

Airaksinen, A., 2018. Tilastokeskus - 5. Big data [Verkkodokumentti]. URL https://www.stat.fi/til/icte/2018/icte_2018_2018-11-30_kat_005_fi.html (Viitattu 25.10.21)

Atwal, H., 2019. Practical DataOps: Delivering Agile Data Science at Scale. Apress, Berkeley, CA, 275 s. ISBN 9781484251034

Basics of ETL Testing with sample queries | Datagaps, 2021. URL https://www.datagaps.com/data-testing-concepts/etl-testing/ (Viitattu 13.8.21).

Bassil, Y., 2012. A Simulation Model for the Waterfall Software Development Life Cycle. International Journal of Engineering &Technology, Vol.2, Nro.5

Big Data: What it is and why it matters | SAS [Verkkodokumentti], URL https://www.sas.com/en_us/insights/big-data/what-is-big-data.html (Viitattu 6.10.21).

Compare Repositories - Open Hub [Verkkodokumentti], URL https://www.openhub.net/repositories/compare (Viitattu 1.10.21).

Ebert, C., Gallardo, G., Hernantes, J., Serrano, N., 2016. DevOps. IEEE Software 33, s.94–100. https://doi.org/10.1109/MS.2016.68

El-Sappagh, S.H.A., Hendawi, A.M.A., El Bastawissy, A.H., 2011. A proposed model for data warehouse ETL processes. Journal of King Saud University - Computer and Information Sciences 23, s.91–104. https://doi.org/10.1016/j.jksuci.2011.05.005

Elgamal, N., Elbastawissy, A., Galal-Edeen, G., 2013. Data Warehouse Testing. Presented at the ACM International Conference Proceeding Series. https://doi.org/10.1145/2457317.2457319

Extract, transform, load [Verkkodokumentti]. Saatavissa: https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/WIKIPEDI/W121031E.pdf (Viitattu: 10.9.2021)

Farmer, D, 2015. [PDF] Speeding ETL Processing in Data Warehouses White Paper [Verkkodokumentti]. URL https://silo.tips/download/speeding-etl-processing-in-data-warehouses-white-paper (Viitattu 8.10.21).

Gandomi, A., Haider, M., 2015. Beyond the hype: Big data concepts, methods, and analytics. International Journal of Information Management 35, s.137–144. https://doi.org/10.1016/j.ijinfomgt.2014.10.007

Gartner Hype Cycle for Data Management Positions Three Technologies in the Innovation Trigger Phase in 2018 [Verkkodokumentti]. URL https://www.gartner.com/en/newsroom/press-releases/2018-09-11-gartner-hype-cycle-for-data-management-positions-three-technologies-in-the-innovation-trigger-phase-in-2018 (Viitattu 2.18.22).

Goar, V., Sarangdevot, S., Tanwar, G., Sharma, D.A., 2010. Improve Performance of Extract, Transform and Load (ETL) in Data Warehouse. International Journal on Computer Science and Engineering 2. s.786–789

Golfarelli, M., Rizzi, S., 2009. Data Warehouse Design: Modern Principles and Methodologies, 1st edition. ed. McGraw-Hill Education, New York. 480 s. ISBN 9780071610391

Goldfedder, J., 2020. Building a Data Integration Team: Skills, Requirements, and Solutions for Designing Integrations, 1st ed. edition. ed. Apress. 267 s. ISBN 9781484256527

Google Trends [Verkkodokumentti]. Google Trends. URL https://trends.google.com/trends/explore?date=all&q=Git,SVN (Viitattu 1.10.21).

Grasse, D., Nelson, G., 2010. Base SAS® vs. SAS® Data Integration Studio: Understanding ETL and the SAS® Tools Used to Support It. Paper presented at the SUGI 31.

Grönfors, M., 2011. Laadullisen tutkimuksen kenttätyömenetelmät. SoFia-Sosiologi-Filosofiapu Vilkka. Hämeenlinna. 153 s. ISBN 9789529300495

Gupta, G.K., 2014. Introduction to data mining with case studies. PHI Learning Pvt. Ltd. 476 s. ISBN 9788120350021

Haavisto, S., 2015. Avoimen datan lyhyt sanakirja [Verkkodokumentti]. Museopro. URL https://www.museopro.fi/fi/nakokulmat.php?aid=14677&k=14295 (Viitattu 23.10.21).

Helsingin Yliopiston tietojenkäsittelytieteen osasto. Git- ja GitHub-sanasto [Verkkodokumentti]. URL https://www.cs.helsinki.fi/u/hisahi/sanastot/git_github.html (Viitattu 28.9.21).

Iyengar, V.S., 2002. Transforming Data to Satisfy Privacy Constraints. s.279–288. https://doi.org/10.1145/775047.775089

Jain, A, 2016. The 5 V's of big data [Verkkodokumentti]. Watson Health Perspectives. URL https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/ (Viitattu 5.10.21).

Kakish, K., Kraft, T., 2012. ETL Evolution for Real-Time Data Warehousing. Proceedings of the Conference on Information Systems Applied Research. New Orleans Louisiana, USA

Kautto, T, 1996. Ohjelmistotestaus ja siinä käytettävät työkalut [Verkkodokumentti]. URL http://www.mit.jyu.fi/opiskelu/seminaarit/ohjelmistotekniikka/testaus/ (Viitattu 1.10.2021).

Kimball, R., Ross, M., 2013. The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd Edition, 3rd edition. ed. Wiley, Indianapolis, IN. 480 s. ISBN 9780071610391

Laney, D, 2001. 3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety [Verkkodokumentti]. pdfcoffee.com. URL https://pdfcoffee.com/ad949-3d-data-management-controlling-data-volume-velocity-and-varietypdf-pdf-free.html (Viitattu 8.10.2021)

Lans, R. van der, 2012. Data Virtualization for Business Intelligence Systems: Revolutionizing Data Integration for Data Warehouses, 1st edition. ed. Morgan Kaufmann, Amsterdam. s.147–176. ISBN 9780123944252

Liikenne- ja viestintäministeriö. 2014. 'Big datan hyödyntäminen' Liikenne- ja viestintäministeriön julkaisuja 20/2014. [verkkojulkaisu]. ISSN 1795-4045. Saatavissa: https://julkaisut.valtioneuvosto.fi/bitstream/handle/10024/77879/Julkaisuja_20-2014.pdf

Lindholm, T. ja ajankohtaisblogit: M., 2021. Avoin data – harrastetoiminnasta julkiseksi palveluksi | Tieto&trendit [Verkkodokumentti]. Saatavissa: https://www2.tilastokeskus.fi/tietotrendit/blogit/2021/avoin-data-harrastetoiminnasta-julkiseksi-palveluksi/ (Viitattu: 7.10.2021).

Malmsten, C.F., 2010. Evolution of Version Control Systems - Comparing CENTRALIZED against DISTRIBUTED Version Control models. Kandidaatintyö. Göteborgin yliopisto. Saatavissa: https://gupea.ub.gu.se/handle/2077/23474 (Viitattu 2.10.2021)

Manyika, J. Chui, M. Brown, B. Bughin, J. Dobbs, R. Roxburgh, Hung Byers, A. 2011. Big Data: The Next Frontier for Innovation, Competition, and Productivity. Saatavissa:

https://www.semanticscholar.org/paper/Big-data%3A-The-next-frontier-for-innovation%2C-and-Manyika/91b63db746becca15090963a8990dfe2b5103799 (Viitattu 7.10.2021).

Marinos, A., Wilde, E., Lu, J., 2010. HTTP database connector (HDBC): RESTful access to relational databases. s. 1157–1158. https://doi.org/10.1145/1772690.1772852

Miller, H., Mork, P., 2013. From Data to Decisions: A Value Chain for Big Data. IT Professional 15, s.57–59. https://doi.org/10.1109/MITP.2013.11

Miller, R., Collins, C., 2002. Acceptance Testing. In: Sensory Evaluation of Food. Food Science Text Series. Springer, New York, NY. https://doi.org/10.1007/978-1-4419-6488-5_14

Mishra, A., Dubey, D., 2013. A Comparative Study of Different Software Development Life Cycle Models in Different Scenarios. International Journal of Advance Research in Computer Science and Management Studies 1, 64–69.

Monica, G, Vignesh, S, Istheyak, N, Prakash, R, Sangar, M, 2019. Data Validation for Nexus Solution using Talend. IJRESM 2, s.599–601.

Moss, L.T., Atre, S., Yourdon, E., 2003. Business Intelligence Roadmap: The Complete Project Lifecycle for Decision-Support Applications, 1st edition. ed. Addison-Wesley Professional, Boston, MA. 576 s. ISBN 9780201784206

Mäkiaho, P., Poranen, T., Seppi, A., 2014. Version control usage in students' software development projects, in: Proceedings of the 15th International Conference on Computer Systems and Technologies, CompSysTech '14. Association for Computing Machinery, New York, NY, USA, s. 452–459. https://doi.org/10.1145/2659532.2659646

Novak, M., Rabuzin, K., 2014. Prototype of a Web ETL Tool. International Journal of Advanced Computer Science and Applications 5. https://doi.org/10.14569/IJACSA.2014.050614

Oguntimilehin, A., Ademola, O., 2014. A Review of Big Data Management, Benefits and Challenges. Journal of Emerging Trends in Computing and Information Sciences 5, s.433–438.

Oliveira, B., Belo, O., 2017. Validating ETL Patterns Feasability using Alloy, in: Proceedings of the 6th International Conference on Data Science, Technology and Applications. Presented at the 6th International Conference on Data Science, Technology and Applications, SCITEPRESS - Science and Technology Publications, Madrid, Spain, s. 200–207. https://doi.org/10.5220/0006428002000207

Patel, M., Patel, D., 2020. Progressive Growth of ETL Tools: A Literature Review of Past to Equip Future. s. 389–398. https://doi.org/10.1007/978-981-15-6014-9_45

Pitkäranta, A. 2010. Laadullisen tutkimuksen tekijälle. Työkirja. Satakunnan AMK. Saatavissa: https://docplayer.fi/2847497-Laadullisen-tutkimuksen-tekijalle.html (Viitattu 19.11.2021)

Phillips, S., Sillito, J., Walker, R., 2011. Branching and merging: an investigation into current version control practices, in: Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE '11. Association for Computing Machinery, New York, NY, USA, pp. 9–15. https://doi.org/10.1145/1984642.1984645

Päivärinta T, 2019. Alustatalousliiketoimintamallin hyödyntäminen uusilla toimialoilla: Monitapaustutkimus: Suomen terveydenhuollon toimiala. Pro Gradu -tutkielma. Turun kauppakorkeakoulu, Turku. Saatavissa: https://urn.fi/URN:NBN:fi-fe201902276508 (Viitattu 16.1.2022)

Rahm, E. & Do, H.H., 2000. Data Cleaning: Problems and Current Approaches. Bulletin of the Technical Committee on Data Engineering, 23(4). s. 3–13. Saatavissa: https://cs.brown.edu/courses/cs227/archives/2017/papers/data-cleaning-IEEE.pdf#page=5 (Viitattu 11.10.2021)

Rahmana, N., Kumarb, N., Rutzc, D., 2016. Managing application compatibility during ETL tools and environment upgrades. Journal of Decision Systems 25, s.136–150. http://dx.doi.org/10.1080/12460125.2016.1138392

Raymond, E, 2011. A History of Version Control [Verkkodokumentti]. URL https://ericsink.com/vcbe/html/history_of_version_control.html#ftn.idp96624 (Viitattu 16.10.21).

Reeves, L., 2009. A Manager's Guide to Data Warehousing, 1st edition. ed. Wiley, Indianapolis, IN. 480 s. ISBN 9780470176382

Schiefer, J., Jeng, J., Bruckner, R., 2003. Real-time workflow audit data integration into data warehouse systems. s. 1697–1706.

Sherman, R., 2014. Business Intelligence Guidebook: From Data Integration to Analytics, 1st edition. ed. Morgan Kaufmann. 550 s. ISBN 9780124114616

Somasundaram, R., 2013. Git: Version control for everyone. Packt Publishing, Birmingham. 180 s. ISBN 9781849517522

Stake, R.E., 2013. Multiple Case Study Analysis. Guilford Press. 457 s. ISBN 9781462512409

Sravanthi, K. & Reddy, T.S., 2015. Applications of Big data in Various Fields. International Journal of Computer Science and Information Technologies, Vol. 6 (5) s. 4629–4632.

Teittinen, V., 2019. Control System Framework automaatiojärjestelmien ohjelmistokehityksessä. AMK-opinnäytetyö, Automaatiotekniikan koulutusohjelma. Kaakkois-Suomen ammattikorkeakoulu.

Tietotekniikan termitalkoot. Big Data. [verkkoaineisto]. Saatavissa: https://termipankki.fi/tepa/fi/haku/big%20data (Viitattu 7.10.2021)

Valli, R., 2018. Ikkunoita tutkimusmetodeihin 2: Näkökulmia aloittelevalle tutkijalle tutkimuksen teoreettisiin lähtökohtiin ja analyysimenetelmiin, 5. ed. PS-kustannus. ISBN: 9789524518253

Vandana, K.V., Sujatha, V., 2013. ETL TESTING IN DATAWAREHOUSING. IJESR 4 Special Issue 01., s.1028–1031

Version Control Systems Popularity in 2016 [Verkkodokumentti]. 2016. RhodeCode. URL https://rhodecode.com/insights/version-control-systems-2016 (Viitattu 1.10.21).

Walrad, C., Strom, D., 2002. The importance of branching models in SCM. Computer 35, 31–38. https://doi.org/10.1109/MC.2002.1033025

Wong, W.E., Horgan, J.R., London, S., Agrawal, H., 1997. A study of effective regression testing in practice, in: Proceedings The Eighth International Symposium on Software Reliability Engineering. Presented at the Proceedings The Eighth International Symposium on Software Reliability Engineering, s. 264–274. https://doi.org/10.1109/ISSRE.1997.630875

Ylijoki, O., Porras, J., 2016. Perspectives to Definition of Big Data: A Mapping Study and Discussion. Journal of Innovation Management 4, s.69–91. https://doi.org/10.24840/2183-0606_004.001_0006

Yulianto, A., 2019. Extract Transform Load (ETL) Process in Distributed Database Academic Data Warehouse. APTIKOM Journal on Computer Science and Information Technologies 4, s.64–71. https://doi.org/10.11591/APTIKOM.J.CSIT.36

Zode, M., 2007. The Evolution of ETL-From Hand-coded ETL to Tool-based ETL.

Zolkifli, N.N., Ngah, A., Deraman, A., 2018. Version Control System: A Review. Procedia Computer Science, The 3rd International Conference on Computer Science and Computational Intelligence (ICCSCI 2018) : Empowering Smart Technology in Digital Era for a Better Life 135, s.408–415. https://doi.org/10.1016/j.procs.2018.08.191

Liite 1: Haastattelurunko

Alkukysymykset

1. Mikä on nimenne ja ammattinimikkeenne?
2. Millaisessa roolissa työskentelette?
3. Kuinka monessa asiakasprojektissa työskentelet?
4. Kuinka monta ihmistä toimii projektissa ETL kehittäjänä tavalla tai toisella?
5. Millainen ETL-kehityksen prosessi on tällä hetkellä alusta loppuun?
    a. Kokonaan uuden kokonaisuuden rakentamisessa?
    b. Olemassa olevan ylläpidossa ja/tai muokkauksessa?

Sisältökysymykset

6. Kuinka montaa ympäristöä käytetään? (esim. Kehitys, Tuotanto, Testi jne.)
    a. Mikä on näiden jokaisen merkitys kehitysprosessissa ja kokonaisuudessa?
    b. Miten versioiden siirtäminen ympäristöstä toiseen tapahtuu ja miten paljon tähän kuluu aikaa?
7. Mitä ETL-työkalua käytetään ja onko kyseisellä työkalulla jotain ominaispiirteitä?
    a. Pystytäänkö samaa tiedostoa/kokonaisuutta kehittämään samaan aikaan ja muutokset yhdistämään, vai pitääkö tiedosto ns. lukita yhdelle kehittäjälle?
    b. Millä tiedostotyypillä ympäristöstä tuotu tiedosto tallennetaan?
8. Hyödynnetäänkö asiakasprojektissanne versionhallintaa? (Jos ei, siirrytään kysymykseen 9.)
    a. Miten kuvailisit nykyistä versionhallinnan toimintakokonaisuutta ja toimintamallia?
    b. Mitä versionhallintajärjestelmää käytetään? Esim. Git tai SVN

   c. Missä funktiossa versionhallintaa pidetään? Esim. enimmäkseen työkalusta riippumattomana varmuuskopiona, ETL kehityksen tukena tms.

   a. Jos käytössä on versiohaaroja mikä on näiden nimeämislogiikka?

   b. Mikä on versionhallinnassa pisin manuaalinen prosessi?

2. Jos asiakasprojektissa ei hyödynnetä erillistä versionhallintaa, onko näin tehty tarkoituksella? Miksi?

   a. Onko versionhallinnan hyödyntämistä mietitty tai suunniteltu asiakasprojektissa?

3. Minkälaisia haittoja tai hyötyjä näet asiakasyritykselle, mikäli versionhallinnan tukena olisi työkalu? (Jos versionhallinta on käytössä) * Kuvailu ideasta

4. Millaiselle asiakkaalle erillinen versionhallinta olisi järkevä ratkaisu?

5. Onko asiakasprojektin versionhallintajärjestelmässä jotain puutteita?

Lopetuskysymykset

1. Mitä hyötyjä versionhallinnan käyttämisestä olisi asiakasyritykselle/kehittäjille? (Jos versionhallintaa ei käytetä)

2. Onko kysymyksistä unohtunut jotain olennaista?