Haaga-Helia
ammattikorkeakoulu Oy

Tietovaraston ETL-prosessin testiautomaatiotyökalun suunnittelu ja toteutus

Panu Schutschkoff

Opinnäytetyö
Tietojenkäsittelyn
koulutusohjelma
2017

Tiivistelmä

Haaga-Helia

Tekijä(t)
Panu Schutschkoff

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Tietovaraston ETL-prosessin testiautomaatiotyökalun suunnittelu ja toteutus

Sivu- ja liitesivumäärä
26 + 2

Opinnäytetyön aiheena on tietovaraston ETL-prosessin testiautomaatiokomponentin toteutusprojekti ja siinä kuvataan projektin kulkua läpi määrittely-, suunnittelu- toteutus- ja testausvaiheiden. Projekti on toteutettu toimeksiantona Aureolis Oy:lle ja sen tuloksena syntyi komponentti, jota hyödyntämällä erilaisten tietovarastointiprojektien automaattitestaus voidaan suorittaa.

Aiemmin yrityksessä ei ole ollut käytössä yleiskäyttöistä työkalua tai toimintamallia automaatiotestauksen toteuttamiseksi tietovarastointiprojekteissa, vaan testaaminen on tapahtunut lähes poikkeuksetta manuaalisesti. Projektin tulosta on tarkoitus hyödyntää erilaisissa projekteissa ja sen avulla voidaan luoda kustannustehokkuutta säästämällä manuaaliseen testaukseen käytetyssä ajassa.

Komponentti toteutettiin yhdistelemällä SQL:a, Microsoft SSIS-integraatiotyökalua ja Robot Frameworkia. Komponentti toimii pääsääntöisesti SQL-kielellä ja sillä tehtyjä proseduureja kutsutaan Robot Frameworkin ja SSIS-pakettien avulla. Komponentin tarkoitus on toimia dynaamisesti riippumatta siitä mikä Microsoft SQL Server -tietokanta on käytössä tai mitä tietoa siellä on. Tämän tarpeen täyttäminen onnistui, sillä komponenttia hyödynnetään käyttämällä erillisiä CSV-muotoisia tiedostoja, joissa testattava taulu on kuvattuna. Testien käyttämiä proseduureja kutsutaan parametreilla, joissa määritellään kaikki tarpeellinen testien suorittamiseksi.

Tietovaraston ETL-latausten testaamisen tuloksena syntyneet onnistuneet ja virheelliset testiajot myös historioidaan, joka mahdollistaa myös testitulosten seurannan ja analysoinnin. Projekti onnistui ja täytti sille annetut vaatimukset, koska sen tuloksena syntyi toimiva testiautomaatiokomponentti, joka ei ole ympäristösidonnainen ja sen avulla saadaan onnistuneesti testattua ETL-prosessin sujuvuus.

Asiasanat
Tietovarastointi, testiautomaatio, ETL, ohjelmistotestaus

Sisällys

1 Johdanto

Tässä opinnäytetyössä käsiteltävä projekti oli toimeksianto Aureolis Oy:lle ja projektin tarkoituksena oli suunnitella ja toteuttaa Microsoft-pohjaisen tietovaraston latauksen validoiva automaatiotestauskomponentti. Tarve komponentin toteuttamiselle syntyi yrityksen luodessa uutta tuote- ja palvelukonseptia, jonka avulla erilaisia tietovarastoprojekteja voidaan toteuttaa kustannustehokkaammin.

Projektin tavoitteena oli saada mahdollisimman helppokäyttöinen ja laajennettavissa oleva komponentti, jonka avulla voidaan säästää aikaa ja työtä ja siten myös kustannuksia tietovarastoprojekteissa. Projektin onnistunut lopputulos on toimiva komponentti, jonka avulla voidaan suorittaa erilaisten tietovarastojen ja niiden tiedoista koottujen raporttien testausta ja validointia. Työn tulosta hyödynnetään useissa tietovarasto- ja raportointiprojektissa ja sen avulla vähennetään manuaaliseen testaukseen kuluvaa aikaa ja kustannuksia.

Testiautomaation tarkoituksena on luoda säästöjä kustannuksissa poistamalla manuaalisesti suoritettavia testejä. Tämä vapauttaa testaajien aikaa suorittamaan vaativampia testitapauksia, joita ei ehkä ole mahdollista edes automatisoida. Säästöt tarkoittavat yritykselle myös kilpailuetua, koska palvelua ja tuotteita voidaan tarjota halvempaan hintaan manuaalisen työn määrän pudotessa.

Aluksi opinnäytetyössä avataan projektin toteutukseen liittyvien asioiden teoriaa loogisessa järjestyksessä tietovarastointi ensin ja tämän jälkeen avataan testiautomaatiota. Tämän jälkeen käydään läpi projektin toteutumista, joka tapahtui ketteriä ja iteratiivisia menetelmiä sekä perinteistä vesiputousmallia yhdistelemällä. Alussa kerron siitä mitä tarvittiin siihen, että projektin toteutusosio ja konkreettisia tuloksia saadaan työn alle. Tämän jälkeen keskitytään komponentin toteutukseen ja testaukseen.

Lopuksi pohdin omia havaintojani omasta oppimisestani, komponentin jatkokehitysideoista ja erilaisista projektin aikana nousseista huomioista.

Opinnäytetyö on suunnattu tietovarastoinnista ja tietojen integraatiosta jo aiempaa kokemusta omaaville tai asiaan valmiiksi perehtyneille.

2 Käsitteistö

Käsite	Kuvaus
Tietovarasto, Data Warehouse	Erilaisten järjestelmien tiedoista koostuva dimensionaaliseen malliin rakennettu tietokanta, jonka tietoja voidaan hyödyntää esimerkiksi raportoinnissa ja analytiikassa.
Data Mart	Otos tietovarastosta, jossa tietovaraston taulujen ja/tai tietojen määrää on rajattu loppukäyttäjien tarpeiden mukaisesti.
Dimensionaalinen malli	Esimerkiksi tähti- tai lumihiutalemalli, käytetään tietovaraston tietokannan tietomallina.
Tähtimalli	Tietokantamalli, jossa jokainen dimensiotaulu on yhteydessä suoraan faktatauluun.
Lumihiutalemalli	Tietokantamalli, jossa dimensiotaulu voi olla yhteydessä joko faktatauluun tai toiseen dimensiotauluun. Yleensä tiedot normalisoidaan 3. normaalimuotoon.
Extract, Transform, Load (ETL)	Prosessi, jonka aikana tiedot siirretään erilaisista lähdejärjestelmistä yhteen tietokantaan ja sen aikana tietoja ja tiedon muotoa yhtenäistetään.
Tietojen harmonisointi	Tietojen muuntaminen samaan esitysmuotoon, esimerkiksi päivämäärien esitysmuodot tai henkilö- ja y-tunnukset.
Faktataulu	Tietokantataulu, joka sisältää yleensä transaktiotason tietoa. Esimerkiksi myyntitapahtuma tai muu operatiivisen järjestelmän tapahtuma.
Dimensiotaulu	Tietokantataulu, joka sisältää faktatauluissa usein toistuvaa tietoa. Esimerkiksi tuote, asiakas tai toimipiste.

Lähdejärjestelmä	Tietojärjestelmä, josta tuodaan tietoa tietovarastoon. Lähdejärjestelmiä voivat olla esimerkiksi toiminnanohjaus-, asiakkuudenhallinta- tai varastojärjestelmä.
Testiautomaatio, Automaatiotestaus	Automatisoitu tietojärjestelmän testaus, joka on ajastettu tai muulla tavalla asetettu suoritettavaksi automaattisesti. Testitapaukset on valmiiksi ohjelmoitu suoritettavaksi, jolloin vältetään manuaalista työtä.
Testitapaus	Yksittäinen käyttäjän tai järjestelmän tekemä suoritus, jonka oikeasta toiminnasta halutaan varmistua.
Robot Framework	Testiautomaatiosovellus, jonka avulla voidaan luoda ja suorittaa testitapauksia automaattisesti.

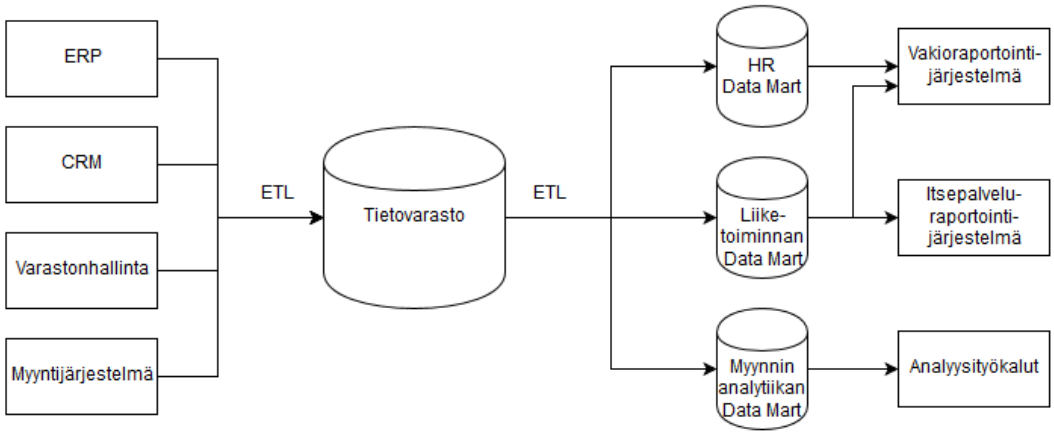
3 Tietovarastointi ja testiautomaatio

Tietovarastot ja niiden päälle rakennettavat Business Intelligence -järjestelmät ovat tällä hetkellä nouseva trendi yritysmaailmassa. Liiketoimintatietoa käsittelevien ja esittävien järjestelmien hyödyntäminen yrityksen päätöksenteossa on yritykselle kannattavaa, koska päätökset pohjautuvat faktatietoon eikä yksittäisten henkilöiden mielipiteisiin (Vercellis, 2009, esipuhe).

Tämän kaltaisten järjestelmien ongelmana voi välillä olla virheet datassa ja sen latauksissa. Näitä ongelmia on selvitetty aiemmin manuaalisesti, joka on työlästä ja aikaa vievää. Automaatiotestauksella saatavat hyödyt ETL-prosessin aikana voivat olla merkittävät ja sen avulla voidaan vapauttaa aikaa muihin tärkeisiin suunnittelu-, kehitys- ja testaustöihin.

3.1 Tietovarastointi

Tietovarastolla tarkoitetaan organisaation käytössä olevaa tietokantaa tai tietokantoja, joissa on liiketoiminnan kannalta merkittävää tietoa. Tietovarastossa sijaitsevaa tietoa hyödyntämällä organisaatiolla on mahdollisuus tutkia liiketoiminnan tuloksia ja analysoida dataa huomattavasti tehokkaammin, kuin poimimalla tieto manuaalisesti eri järjestelmistä, yhdistämällä ne ja lopuksi tekemällä analyysiä tai johtopäätöksiä tiedosta. Tietoa hyödyntämällä organisaatio kykenee tekemään liiketoimintaa tukevia tai tehostavia päätöksiä, sekä löytämään erilaisia ajasta, tuotteesta tai paikasta liittyviä trendejä. Toinen merkittävä hyöty on päätöksenteon pohjautuminen faktatietoon, eikä yksittäisten henkilöiden mielipiteisiin tai muistiin. (Vercellis, 2009, esipuhe)



(Kuva 1 Esimerkki yrityksen tietovarastosta ja sen ympärille rakennetusta järjestelmästä)

Tietovarastossa sijaitsevan tiedon tulee olla eheää ja yhtenäistä, eli eri lähdejärjestelmistä tulevien samaan toimintoon, tuotteeseen, toimipisteeseen tai muuhun vastaavaan liittyvien tietojen tulee päätyä tietovarastossa samaan muotoon. Tällä tavalla tiedon laatu pysyy hyvänä ja yhtenäisenä tietovarastossa. Laadukasta tietoa sisältävästä tietovarastosta on helppo koota erilaisia faktatietoon pohjautuvia suorituskykymittareita, esimerkiksi myynnistä tuote- tai tuoteryhmätasolla sekä myynnistä toimipisteittäin. (Kimball, Reeves, Ross & Thornwaite 2002, 11-12). Kuvan 1 vasemmassa reunassa on kuvattu esimerkkejä erilaisia järjestelmiä, joiden tiedoista tietovarasto voi koostua. Lähdejärjestelmiä voivat olla esimerkiksi toiminnanohjausjärjestelmät, asiakkuudenhallintajärjestelmät ja erilaiset operatiiviset järjestelmät, joita hyödynnetään yrityksen liiketoiminnassa. Näiden järjestelmien tiedot kootaan tietovarastoon niin, että asiakastieto esimerkiksi toiminnanohjausjärjestelmässä vastaa samaa asiakastietoa asiakkuudenhallinta- ja myyntijärjestelmässä.

Tietovaraston datasta voidaan myös tehdä Data martteja. Data mart on otos tietovarastosta, johon on valittu mukaan tietyt taulut ja kokonaisuudet käyttötarpeiden mukaan. Yrityksen tietovarasto saattaa sisältää kaiken tiedon henkilöstöhallinnon tiedoista myyntitietoihin, eikä kaikki tieto ole jokaisen tietovarastoa hyödyntävän henkilön tehtävien kannalta tarpeellista, joten useimmissa tapauksissa on syytä rajata tietoa, joka parantaa suorituskykyä ja nopeuttaa haluttujen tietojen hakemista ja laskemista. (Silvers 2008, 129-130) Kuvassa 1 Data martteja on kuvattu kolme kappaletta, joista jokaisella on oma käyttötarkoituksensa. Esimerkiksi henkilöstöhallinnon data martissa voi olla tietoja ainoastaan yrityksen työntekijöistä, palkanmaksusta ja henkilöstökyselyiden tuloksista. Erilaisilla otoksilla tietovarastosta saadaan loogisesti toisiinsa liittyviä tiiviitä aihekokonaisuuksia, jolloin tietojen käyttäminen ja yhdistäminen toisiinsa on huomattavasti helpompaa loppukäyttäjälle.

Tieto viedään tietovarastosta tai data martista raportointia varten valittuun malliin, jossa tiedolle suoritetaan valmiiksi laskentoja, aggregointia ja ryhmittelyä, joka nopeuttaa tietojen käyttöä. Liiketoiminnan kannalta tärkein malli on MOLAP, eli Multidimensional On-Line Analytical Processing. Jossa dataa voidaan tarkastella ja pilkkoa haluttujen dimensioiden tietojen mukaan. Tiedot, joilla dataa ryhmitellään ja suodatetaan, ovat pääsääntöisesti dimensiotietoa. (Paredes 2009, 12-16)

Hyödyt tietovaraston rakentamisesta jäävät pieneksi, mikäli sitä käyttävät liiketoimintayksiköt tai henkilöt eivät ole mukana sen suunnittelussa ja määrittelyssä, koska edellä mainitut pystyvät kertomaan yksityiskohtaisempia ja oikeita tarpeita tiedon hyödyntämiseen liittyen. Mikäli tietovarasto kykenee täyttämään käyttäjiensä

liiketoimintatiedon hakuun ja käsittelyyn liittyvät tarpeet hyvin, on siitä saatu hyöty huomattavasti suurempi. (Atre, Moss 2003, 37-39)

3.1.1 Fakta- ja dimensiotaulut

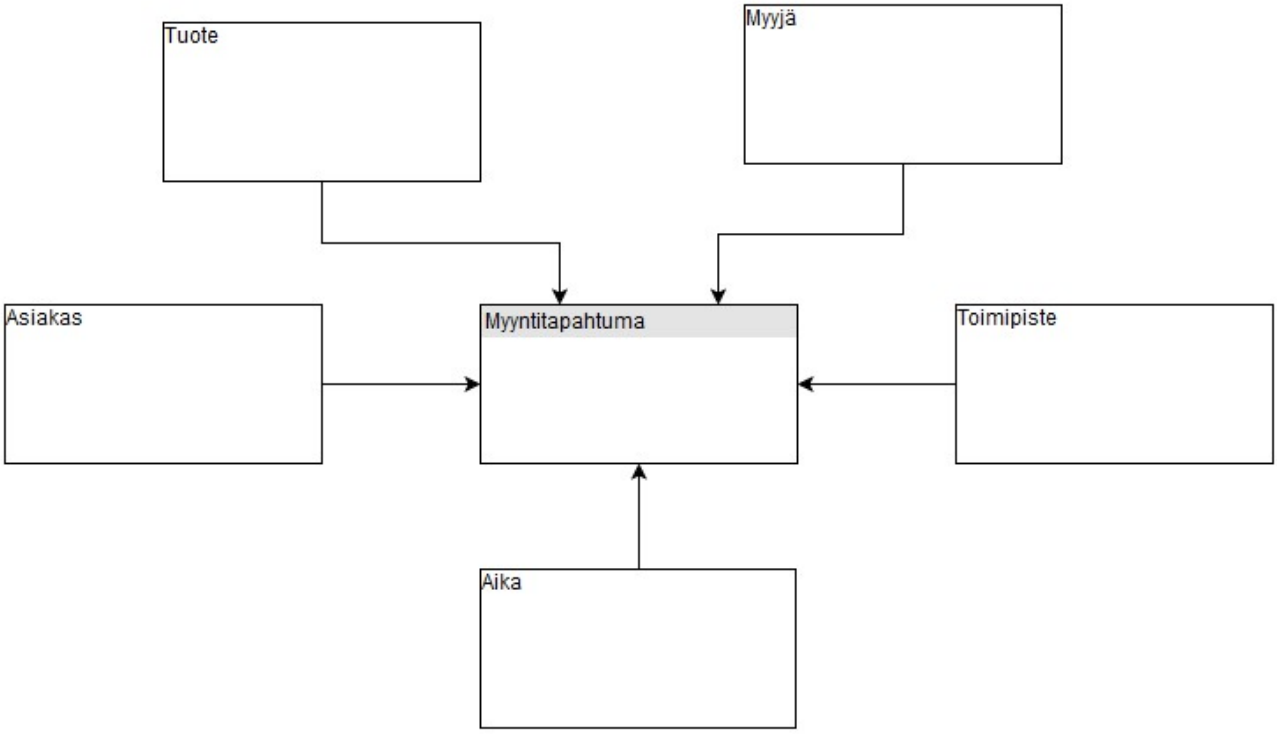
Tietovarastossa data ladataan joko fakta- tai dimensiotauluun, joista molemmilla on omanlaisensa käyttötarkoitus. Pelkästään fakta- tai dimensiotaulut eivät riitä muodostamaan tietovarastoa, josta voi saada merkittävää liiketoiminnallista hyötyä. Faktatauluihin ladataan nimensä mukaisesti faktatietoa, jonka halutaan useimmiten olevan mahdollisimman atomista eli mahdollisimman yksityiskohtaiseen muotoon pilkottuna. Tieto voidaan pilkkoa esimerkiksi yksittäisen myyntitapahtuman tasolle. (Silvers 2008, 4.) Näin tietovaraston dataa voidaan hyödyntää yksityiskohtaisemmin ja sen pohjalta tehtävä analyysi on luotettavampaa. (Kimball ym., 2002, 86.) Kuitenkin on mahdollista, että tietoja viedään tietovarastoon summattuna. Tätä vaihtoehto, mikäli yksityiskohtaisempia tietoja ei ole tarvetta käsitellä.

Faktataulut sisältävät yksilöityä ja yksityiskohtaista tietoa liiketoiminnalle keskeisestä toistuvasta tapahtumasta, joita halutaan tarkastella, vertailla tai analysoida, kuten myyntitapahtumasta. Niihin tallennettavan datan pohjalta pyritään täyttämään kyseisiä tauluja hyödyntävien käyttäjäryhmien tarpeet. Tarpeita voivat olla esimerkiksi liikevaihdon laskeminen, katetuottoprosentin muodostaminen tai asiakasmäärän koostaminen. Tämän lisäksi tietoa halutaan usein rajata, suodattaa tai ryhmitellä jonkin lisätiedon perusteella, vaikkapa ajan, paikan tai myydyn tuotteen mukaan. Edellä mainitut tiedot ovat tietovarastossa dimensiodataa ja ne voivat toistua useissa faktataulun riveissä. Esimerkkinä voidaan käyttää esimerkiksi myyntitapahtumaa. Faktataulussa jokainen rivi on uniikki myyntitapahtuma, joka kertoo, että tietty tuote on myyty jollakin hinnalla tietyssä toimipisteessä. Dimensiotauluissa on siis faktatauluissa toistuvasti esiintyvää dataa kun taas faktataulujen jokaisen rivin tulee olla uniikki. (Kimball ym., 2002, 398.)

3.1.2 Tähtimalli

Yksinkertaisin tapa tietovaraston tietomalliksi on tähtimalli (eng. star schema). Tähtimallilla tarkoitetaan tietomallia, jossa on yksi tai useampia faktatauluja, sekä dimensiotauluja jotka ovat viiteavaimella suoraan yhteydessä faktatauluun tai faktatauluihin. Koska tähtimallin faktataulut sisältävät usein paljon dataa, liitokset taulujen välillä usein kuormittavat tietokantapalvelinta merkittävästi. Tämän vuoksi faktatauluja denormalisoidaan, jolloin liitoksia taulujen välillä ei tarvitse tehdä. Denormalisointi tehdään käyttötarpeiden mukaan,

eli tiedon ryhmittelyyn tai suodatukseen käytettäviä sarakkeita voidaan myös jättää denormalisoimatta.



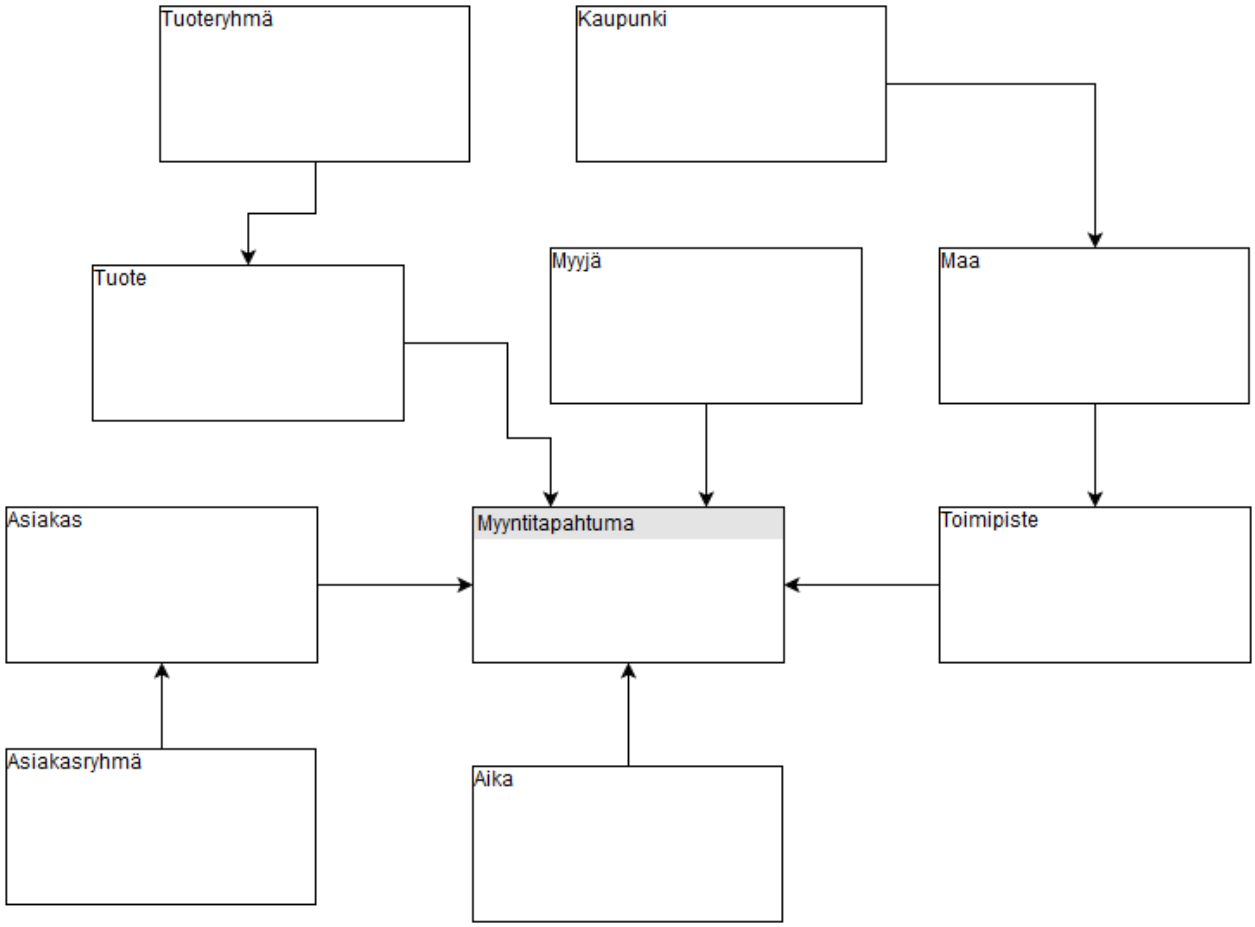
(Kuva 2 Esimerkkikaavio tähtimallista)

Tähtimallia käytettäessä on tärkeää tietää mitä lukuja tietovarastosta halutaan saada näkyviin ja minkälaisen tiedon mukaan niitä halutaan ryhmitellä, suodattaa ja aggregoida. Kun halutaan tarkastella esimerkiksi yrityksen myyntiä, niin pelkästään myynnin määrä ei riitä vaan myyntiä halutaan tarkastella esimerkiksi ajan, paikan tai myydyn tuotteen mukaan. (Silvers 2008, 4.) Kuvassa 2 on esitetty tähtimallin rakennetta, joka on tietovarastossa olevista tiedoista riippumatta aina samanlainen. Yleensä tietomalli muutetaan tähtimallin mukaiseksi viimeistään data martissa, jolloin tietojen hyödyntäminen on loppukäyttäjälle yksinkertaisempaa.

3.1.3 Lumihiutalemalli

Lumihiutalemalli (eng. snowflake schema) on toinen vaihtoehto tietovaraston tietomallin toteuttamiselle. Toisin kuin tähtimallissa, lumihiutalemallin dimensiotaulut ovat normalisoituja. Dimensiotaulujen normalisoiminen vaikeuttaa joidenkin kyselyiden

tekemistä, mutta tietojen normaalimuoto nopeuttaa kyselyiden tekemistä ja käyttö on nopeampaa.

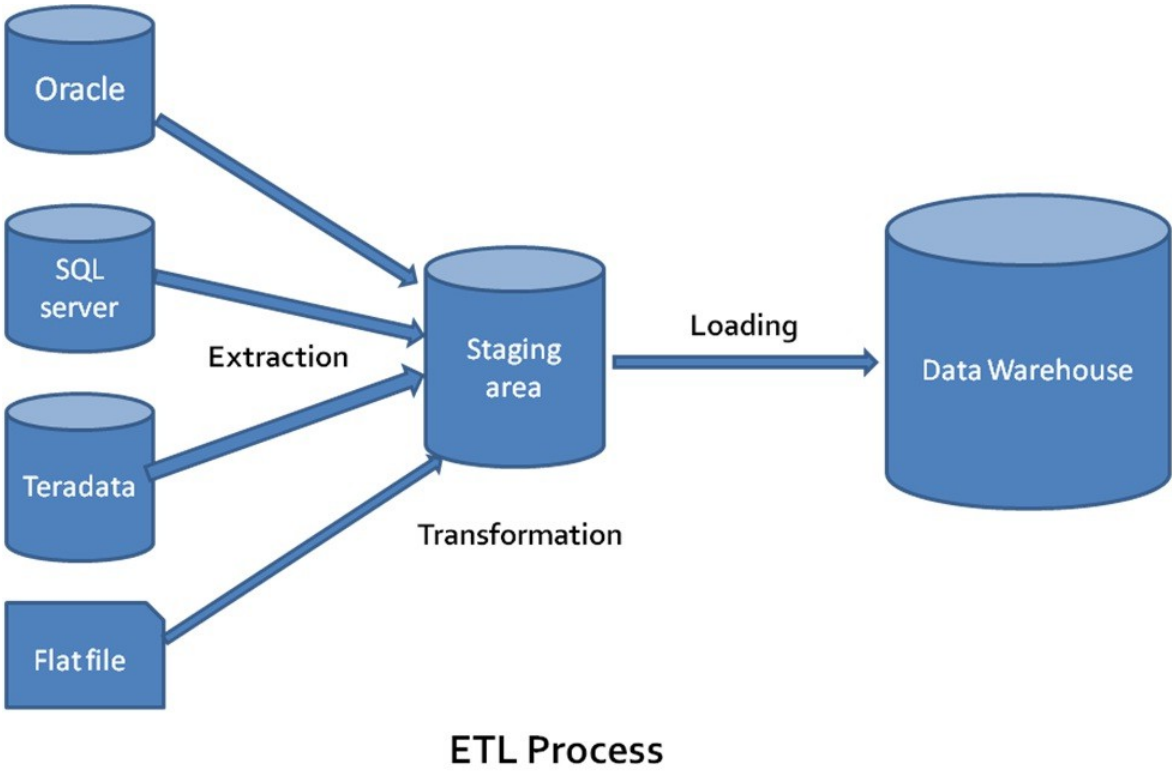


(Kuva 3 Esimerkkikaavio lumihiutalemallista)

Tietoja normalisoimalla voidaan helpommin olla varmoja datan eheydestä, koska faktataulun tieto viittaa aina dimensiotaulun tietoon. Tämä myös aiheuttaa sen, että lumihiutalemallia käyttämällä tietovarasto vie vähemmän levytilaa, koska samaa tietoa ei monisteta useaan eri tauluun. (Ponniah 2004, 237.) Kuvassa kolme on kuvattu lumihiutalemallia ja siitä voidaan havaita, että ainoastaan suoraan faktataulun tietoihin liittyvä dimensiotieto, esimerkiksi tuotetieto, on kytköksissä faktatauluun. Dimensiotietoa täydentävät tiedot ovat kytkettynä toisiin dimensioihin, esimerkiksi tuotetietoa voivat täydentää tuoteryhmä tai toimittaja.

3.1.4 Extract, Transform, Load (ETL)

Lähdejärjestelmistä siirrettävät tiedot on käsiteltävä ja siirrettävä tasaisin väliajoin tietovarastoon. Tämä prosessi on poikkeuksetta automatisoitu ja siitä käytetään nimitystä Extract, Transform, Load. ETL-prosessin toteuttamiseksi on olemassa useita erilaisia työkaluja ja ne ovat pääsääntöisesti riippumattomia taustalla olevasta teknologiasta.



(Kuva 4. ETL-prosessin vaiheet, Tortorella 2015)

Tietovaraston data haetaan yhdestä tai useammasta tietolähteestä. Tietolähteenä useimmissa tietovarastoissa on operatiivinen järjestelmä tai useampia operatiivisia järjestelmiä, kuten toiminnanohjausjärjestelmiä, asiakkuudenhallintajärjestelmä tai jokin liiketoimintatietoa tallentava sovellus. Kuvan 4 vasemmassa reunassa ja keskellä on havainnollistettu, kuinka tietolähteistä saatavat tiedot ladataan Staging-alueen tietokantaan, johon on tarkoituksena koota kaikista tietolähteistä saatava tietovarastoon ladattava data. (Kimball ym., 2002, 11-12)

Erilaisista tietolähteistä haettava data ei ole yhtenäisessä muodossa, joten tiedot tuodaan Staging-alueelle siinä muodossa kun ne saadaan lähdejärjestelmistä. Tietojen siirtoa Staging-alueelle kutsutaan Extract-vaiheeksi. Kun data on ladattu yhteen paikkaan,

voidaan sitä alkaa käsittelemään ja muuntamaan yhtenäiseksi käyttötarkoitusten mukaisesti.

Kun tietovaraston käyttöön tarvittava data on koottu yhteen paikkaan, tulee se puhdistaa ja yhtenäistää. Tätä vaihetta kutsutaan Transform-vaiheeksi. Tämä vaihe toteutetaan ennen tietojen tallentamista tietovarastoon, kuten kuvassa 4 on kuvattu. Tiedon yhtenäistämisellä eli harmonisoinnilla tarkoitetaan vaihetta, jossa eri järjestelmissä olevat samaa asiaa koskevat tiedot muunnetaan samaan muotoon. Tällaisia tietoja voivat esimerkiksi olla myyjä, toimipiste, myyntipäivä tai tuote. Viimeistään muuntamisvaihetta aloitettaessa, tulee sitä varten määritellä taulut, joissa tiedon tulee sijaita, sekä taulujen sarakkeet ja sarakkeiden tietotyypit. Näin toteutusvaiheessa tiedetään, minkälaiseen muotoon tiedot tulee yhtenäistää ja tietovaraston datan eheys varmistuu.

Transform-vaiheessa voidaan tehdä myös muutoksia dataan erilaisten liiketoiminnan luomien käsittelysääntöjen mukaan tietovarastokohtaisesti. Kahdessa eri järjestelmässä voi olla sama ohjaustieto, kuten myyntitapahtumaan liittyvä tieto onko transaktioon liittyvä tuote ostettu vai hyvitetty tai muuta vastaavaa, mutta ohjaustieto on ilmaistu eri tavalla molemmissa järjestelmissä. Yleensä transform-vaiheessa käsitellään tämän kaltaiset tilanteet luomalla käsittelysääntö näille tiedoille, jotta data on yhtenäistä kun se ladataan tietovarastoon. (Atre & Moss, 2003, 221-222)

Kun tietovarastoon ladattavan datan käsittelyt tietojen yhtenäistämiseksi on luotu, ladataan tiedot tietovarastoon. Tätä vaihetta kutsutaan Load-vaiheeksi ja se on tietovaraston tietojen lataamisen ETL-prosessin viimeinen vaihe. Tietovaraston pohjana on tietokanta, jonka tietomallin tavoitteena on tehdä tietovaraston datasta helposti saatavaa ja yhdisteltävää liiketoiminnan tarpeiden mukaisesti. (Kimball ym., 2002, 10.) Mikäli tietoja siirretään tietovarastosta data marteihin, voidaan tällekin välille tehdä oma ETL-prosessinsa, jossa esimerkiksi suodatetaan ja muunnetaan tietoja vastaamaan käyttäjien tarpeita paremmin.

3.1.5 Tietovaraston hyödyntäminen

Tietovaraston avulla organisaatio voi koota käyttämiensä tietojärjestelmien datan yhteen paikkaan ja yhdistää eri järjestelmien tietoa, kunhan ne on liitetty toisiinsa oikein. Helposti saatavilla olevaa ja hyödynnettävää tietoa käyttäen on helpompi luoda erilaisia raportteja ja koostaa toisistaan riippuvaa tietoa. Tämän avulla liiketoiminnan tuloksia ja trendejä voidaan seurata ja analysoida ja tämän ansiosta havaittuihin asioihin voidaan reagoida. Hyvin suunniteltu tietovarasto tarjoaa loppukäyttäjälle mahdollisuuden käyttää juuri sillä

hetkellä tarvitsemaansa tietoa yhdestä lähteestä, mikä helpottaa ja tehostaa raporttien ja analyysin tuottamista.

Yrityksen kannalta tiedon saaminen reaaliaikaisesti eheässä muodossa on merkittävä asia, koska tiedon perusteella saadaan kriittisiä tietoja liiketoimintaan liittyen, joka mahdollistaa nopeankin reagoimisen mahdollisiin muutoksiin liiketoiminnassa. Tämän lisäksi tiedon pohjalta voidaan tehdä analyysiä ja johtopäätöksiä sekä löytää erilaisia trendejä. (Serra 2013)

3.2 Ohjelmistotestaus

Testaamisella tarkoitetaan jonkin kokonaisuuden tai osan laadunvarmistusta, eli sen avulla pyritään löytämään mahdollisia virheitä yksittäisestä toiminnallisuudesta tai kokonaisuudesta. Testausta voidaan suorittaa sekä manuaalisesti, eli joku henkilö toteuttaa käsin testejä erilaisten testitapausten pohjalta, tai automaattisesti, jolloin esimääriteltyjä testitapauksia suoritetaan tietokoneella automaattisesti. (Rainardi 2007, 480-482)

Ohjelmistotestauksessa testauksen tavoitteena on löytää virheitä toteutetuista toiminnallisuuksista sekä mitata ohjelmiston laatua. (Tuovinen 2013) Tietovaraston testaamisessa on huomioitava, että loppukäyttäjän vaatimukset eroavat perinteisen ohjelmiston vaatimuksista. Ohjelmistoissa on usein yksittäisiä pieniä toiminnallisuuksia etenkin käyttöliittymän puolella, joiden tulee toimia oikein. Tietovaraston testauksessa tarkastellaan ainoastaan datan eheyttä ja oikeellisuutta, jotka loppukäyttäjä näkee tietovarastoa hyödyntävän raportin kautta.

Tietovaraston testaamisessa on kaksi kriittistä osa-aluetta, joiden testaaminen on erittäin tärkeää. Nämä kohdat ovat ETL-prosessin Transform-vaihe sekä kuinka tiedon käytettävyys vastaa liiketoiminnan tarpeita. Mikäli Transform-vaiheessa on käytetty virheellisiä käsittelysääntöjä tai joitakin käsittelyjä ei ole tehty, ei tietovarastossa oleva data ole eheää, koska siinä on virheitä. Mikäli data on virheellistä, ei se näytä oikeita tietoja, joka pahimmassa tapauksessa voi saada tietovarastoa päätöksenteossa hyödyntävän käyttäjän tekemään virheellisen päätöksen. Mikäli tietovaraston dataa ei voi yhdistellä vaatimusten mukaisesti, ei käyttäjä voi hyödyntää sen dataa haluamallansa tavalla. Tämän seurauksena liiketoiminnalliset hyödyt jäävät mahdollisesti tai jopa kokonaan saavuttamatta.

Tietovaraston automaattitestaaminen eroaa perinteisestä ohjelmistotestauksesta merkittävästi, koska yksittäisiä ominaisuuksia on huomattavasti haastavampaa testata läpikotaisin. Ohjelmistotestauksessa apuna käytetään yksikkötestejä, joilla testataan yksittäisen toiminnallisuuden toimintaa, kun taas tietovaraston testaamisessa painopisteenä keskitytään kokonaisuuksiin, joissa halutaan varmistua tiedon eheydestä ja oikeellisuudesta. Tietovaraston testauksessa merkittävä ero on myös testidata. Testausta suoritettaessa kaikkia testattavia ETL-latauksia varten tarvitaan testidataa, jotta voidaan varmistua erilaisten käsittelyiden toimivuudesta. (Levy & Oates 2007)

3.2.1 Testausmenetelmät

Ohjelmistotestaukseen liittyen on olemassa erilaisia testausmenetelmiä, joita voidaan käyttää testaamisessa tilannekohtaisesti. Black-box testaaminen on testausmenetelmä, jossa testaajan luomat testitapaukset perustuvat määrittelyihin ja muihin dokumentteihin, joiden pohjalta toteutus on tehty. White-box testaamisessa testaajalla on pääsy lähdekoodiin ja toimintalogiikkaan yksityiskohtaisesti, eli kyseinen menetelmä on päinvastainen black-box testaukseen nähden. Edellä mainittujen lisäksi on olemassa grey-box menetelmä, joka on yhdistelmä black-box ja white-box testausta, eli siinä testaaja voi tapauskohtaisesti hyödyntää esimerkiksi enemmän lähdekoodia ja toimintalogiikkaa mikäli testitapaus niin vaatii. (Räsänen 2013)

Tietovarastotestauksessa on välttämätöntä, että testaaja tuntee ETL-prosessin käsittelylogiikan sekä lähtö- ja kohdetaulut, joten esimerkiksi black-box testaus ei ole kovinkaan hyvä menetelmä kun testataan ETL-prosessin yksittäisiä latauksia. ETL-prosessia testatessa grey-box on tehokkain menetelmä.

3.2.2 Testiautomaatio

Testiautomaatiolla tarkoitetaan ohjelmiston testausprosessien automatisointia. Automaatiotestausta hyödynnetään useimmiten regressiotestauksessa, eli sen avulla tarkastetaan mahdollisia ilmenneitä puutteita jo valmiiksi todetussa ominaisuudessa, koska uudet luodut toiminnot voivat vaikuttaa niihin. Testiautomaation suurin hyöty on manuaalisen työn vähentäminen, mutta se ei kuitenkaan pysty korvaamaan sitä kokonaan. Tällä tavalla testaajien työmäärää voidaan kohdentaa paremmin osa-alueille, joiden testausta ei voida automatisoida.

Automaatiotestauksen käyttöönotto ja valmistelu vaativat alussa huomattavasti enemmän resursseja kuin manuaalinen testaus, koska kaikki testidata on luotava ja testitapaukset automatisoitava. Hyödyt kuitenkin ilmenevät, kun samoja tapauksia testataan aina uudelleen esimerkiksi uuden version julkaisun yhteydessä. Automaattitestin osalta riittää, että se luodaan kerran ja suoritetaan milloin tahansa niin monta kertaa kuin on tarvetta. Manuaalisesti toteutettu testaus vie kuitenkin saman määrän aikaa testaajalta joka kerta, kun kyseinen testi halutaan toteuttaa. Kustannuustehokkuus kasvaa sitä mukaa, mitä useammin testejä suoritetaan. Esimerkiksi testien automatisointiin kuluu 15 tuntia ja niiden manuaaliseen suorittamiseen 3 tuntia. Tämänlainen testitapaus maksaa itsensä takaisin jo viiden ajon jälkeen. Usein testitapauksia on vähintään useita kymmeniä tai satoja, jolloin ajan säästöt ovat merkittäviä. (Dustin, Rashka & Paul 2008, 32-35)

3.2.3 Robot Framework

Robot Framework on avoimen lähdekoodin Pythonilla toteutettu testiautomaatiotyökalu, joka on laajennettavissa käyttäjien tarpeiden mukaan Python- tai Java-pohjaisilla kirjastoilla. Testattavat kokonaisuudet eli test suitet koostuvat useammasta testitapauksesta, jotka on määritelty testitapaukset sisältävään tiedostoon. Testitapausten kriteerit määritellään avainsanoilla ja parametreilla. Testitapaukset ovat yksinkertaisia toteuttaa, koska avainsanat ovat selkokielisiä.

Testiajo käynnistetään komentorivin kautta ja kun se on suoritettu, toteuttaa Robot Framework raportin testien suorituksesta, jolloin käyttäjä näkee välittömästi mitkä testit ovat onnistuneet ja mitkä epäonnistuneet. Etenkin regressiotestaamisen automatisoimiseen, eli kaikkien testitapausten suorittaminen ja muutosten kokonaisvaikutuksen havainnointiin liittyen, Robot Framework on hyödyllinen työkalu laadunvarmistuksen tueksi. Se vähentää huomattavasti manuaalisesti suoritettavaa testausta, mutta ei kuitenkaan kokonaan poista tarvetta suorittaa sitä.

3.2.4 Tietojen integrointi

Tietojen integroinnilla tarkoitetaan tietojen yhdistämistä ja muuntamista samanmuotoiseksi eli harmonisointia ja keskittämistä yhteen paikkaan, esimerkiksi yrityksen tietovarastoon. Yrityksellä on usein erilaisia tietojärjestelmiä joiden tiedot eivät ole yhteydessä keskenään, jolloin näiden tietojen yhdisteleminen on työlästä ja etenkin virhealtista. Erilaisia tietojärjestelmiä voivat olla esimerkiksi toiminnanohjausjärjestelmät,

asiakkuudenhallintajärjestelmät, HR-järjestelmät ja liiketoimintajärjestelmät. Järjestelmien määrä riippuu tietenkin yrityksestä, mutta mitä enemmän niitä on, sitä haastavampaa tietojen yhdisteleminen on.

Tietojen integrointi voi tapahtua esimerkiksi erillisellä ETL-sovelluksella, jonka avulla tietoja voidaan koota eri lähdejärjestelmistä yhteen tietovarastoon graafisen käyttöliittymän avulla. Tässä projektissa ETL-työkaluna käytettiin Microsoftin SQL Server Integration Services (SSIS) -työkalua. Muita vaihtoehtoja ovat esimerkiksi Informatica tai SAS Data Integration Studio.

Ennen kuin dataa eri järjestelmien välillä voidaan integroida, on tärkeää mallintaa tietovarasto hyvin. Näin varmistutaan siltä, ettei samaa tietoa ole kahdessa paikassa ja myös siitä, että tietojen tietotyypit ovat oikein. Tämän lisäksi on välttämätöntä päättää missä muodossa tieto tallennetaan ja esitetään. Esimerkiksi onko asiakkaan nimi tietokantataulussa yhdessä sarakkeessa vai onko se jaettu etunimeen ja sukunimeen. Eri järjestelmissä nämä tiedot voivat olla eri tavalla tallennettuna, jolloin käsittelyiden on harmonisoitava tiedot tietovarastoon. (Doan, Halvey & Ives 2012, 22-23)

4 Komponentin suunnittelu ja toteutus

Projektin tarkoituksena oli toteuttaa dynaaminen testiautomaatiokomponentti, tai vähintään kehys testiautomaatiokomponentista jonka jatkokehitys viedään loppuun myöhemmin. Komponentin vaatimuksena oli ennen projektin aloittamista, että sitä voidaan hyödyntää tietovaraston ETL-prosessin testaamisessa ja validoinnissa yhdessä Aureolis Oy:n projektissa, mutta tämän lisäksi se tulee olla mahdollista ottaa käyttöön myös muissa projekteissa. Komponentin pohjimmaisena tarkoituksena on tuottaa säästöjä testaukseen käytettävässä ajassa ja siten myös luonnollisesti testaukseen liittyvissä kustannuksissa. Säästöt mahdollistavat toimeksiantajalle alemman hinnan erilaisissa tietovarastoprojektien toteutuksissa ja tuovat siten kilpailuetua yritykselle.

Vaatimuksena projektissa toteutettavalle komponentille oli riippumattomuus datasta ja tietovarastosta, eli komponentin toimintalogiikan täytyy toimia oikein missä tahansa Microsoft-pohjaisessa SQL-tietokannassa minkä tahansa datan kanssa. Mitä pienemmällä vaivalla testaaja tai kehittäjä pystyy luomaan dataa testien toimintaa varten, niin sitä suurempi hyöty komponentista on. Samalla kuitenkin kehitystyössä pidettiin tärkeänä, että testaus kattaa mahdollisimman suuren osan ETL-prosessista ja tämän lisäksi myös tuli huomioida testien luotettavuus.

Komponentin kehitystyö tapahtui neljässä eri vaiheessa, jotka olivat määrittely, suunnittelu, toteutus ja testaus. Näistä toteutusta ja testausta pyrittiin käymään läpi iteratiivisesti niin, että ominaisuuden ollessa valmis, se testattiin ja siitä saatiin palaute. Palautteen avulla voitiin korjata siinä ilmenevät virheet tai tehdä muita mahdollisia parannuksia käytettävyyteen tai johonkin ominaisuuteen liittyen. Tarvittaessa kuitenkin pidettiin mahdollisuus auki sille, että määrittelyt ja suunnitelma muuttuvat projektin aikana, mutta tärkeimmät vaatimukset pyrittiin saamaan selville heti projektin alkuvaiheiden aikana.

Määrittely ja suunnittelu toteutettiin yhdessä projektin Scrum-tiimin kanssa, jotta komponentin mahdollisia käyttötarkoituksia tai siihen tarpeellisia ominaisuuksia saatiin koottua mahdollisimman kattavasti. Komponenttia varten koottiin paljon erilaisia ominaisuuksia mitä haluttiin mukaan komponenttiin, mutta ensimmäisessä vaiheessa, eli tämän opinnäytetyön sisältönä, tarkoituksena oli saada toimiva ja dynaaminen komponentti mahdollisimman yksinkertaisilla, mutta toimivilla, toiminnallisuuksilla.

4.1 Määrittelyvaihe

Määrittelyä alettiin toteuttamaan aluksi yhdessä scrum-tiimin kanssa ja vaiheen tarkoituksena oli saada koottua erilaisten potentiaalisten käyttäjien tarpeita, odotuksia ja mielipiteitä heidän tarvitsemaansa testiautomaatiotyökaluun liittyen. Suurimmat toiveet tulevilta käyttäjiltä tässä vaiheessa olivat mahdollisimman pieni työmäärä testaajalle, dynaamisuus, eli riippumattomuus järjestelmästä ja datasta, sekä helppokäyttöisyys.

Kun potentiaalisten loppukäyttäjien mielipiteet ja ajatukset oli kerätty, aloimme pohtimaan ja suunnittelemaan minkälaista komponenttia käytännössä tarvitaan, jotta testaamiseen liittyvät kattavuusvaatimukset täyttyvät. Projektin alkuvaiheessa päätettiin, että uusien kehitysideoiden noustessa ne arvioidaan ja priorisoidaan tapauskohtaisesti ennen kuin päätetään niiden ottamisesta mukaan kehitystyöhön, koska projektille varattu aika tiedettiin erittäin rajalliseksi.

Projektin haasteena tässä vaiheessa oli tarkempien vaatimusten puute ja kehitystiimille annettiin melko vapaat kädet ominaisuuksien osalta. Ajalliset resurssit olivat melko tiukat, joten suunnittelussa pyrittiin keskittymään ainoastaan tärkeimpiin ominaisuuksiin ja komponentin jatkokehitysmahdollisuuksiin.

4.2 Suunnittelu

Kehitystiimin koottua määrittelyt ja suunnitelman komponentin tarkemmista käyttötarkoituksista, ryhdyttiin kartoittamaan komponentin mahdollisia toteutustapoja. Tietoa testiautomaation toteutuksesta tietovaraston tietojen lataukselle haettiin erilaisista lähteistä. Lähteinä käytettiin eri yritysten tuotteita ja erilaisia avoimen lähdekoodin toteutuksia. Tässä ongelmaksi muodostui se, että olemassa olevia ratkaisuja oli olemassa erittäin vähän, eikä esimerkiksi avoimen lähdekoodin työkaluja tai kirjastoja tähän liittyen ollut suunnitteluhetkellä juurikaan olemassa. Valtaosa tietovarastoon ja ETL-prosessiin liittyvistä testiautomaatiotyökaluista on myös maksullisia ja poikkeuksetta niiden hintaluokka nousi tuhansiin euroihin lisenssiä kohti. Lisenssien hankintahintojen ollessa korkeat, jäisi toimeksiantajan saama hyöty pienemmäksi tai taloudellisesti tappiolliseksi. Ongelmana valmiissa tuotteissa tätä käyttötarkoitusta varten oli myös niiden laajat toiminnallisuudet, jotka usein saattavat jäädä kokonaan ilman käyttöä. Tämä aiheuttaa sen, että maksetaan turhaan kalliita lisenssejä, jolloin oman komponentin suunnittelu ja toteutus on perusteltua ja taloudellisesti todennäköisesti kannattavampaa.

Ensimmäinen toteutustapa jota projektissa suunniteltiin, oli C#-pohjainen komponentti, jonka toiminta perustuu käyttäjän antamiin tietoihin sarake- ja taulukohtaisesti.

Komponentin tarkoitus oli toimia omana SSIS-pakettinaan, jolloin sen ajamisen voi ajastaa testipalvelimella ja jättää ottamatta käyttöön tuotantoympäristössä. Käyttäjän oli tarkoitus syöttää esimääriteltyyn Excel-pohjaan tietojen lähde- ja kohdetaulut, sekä molemmista tauluista sarake, jossa automaattitestissä testattavan latauksen tieto on. Tämän lisäksi annetaan mahdollinen käsittelysääntö tai operaatio, jolla tieto muunnetaan, rajataan tai käsitellään ETL-prosessin aikana. Näitä ovat esimerkiksi dimensiotiedon liittäminen faktatauluun tai tiettyjen arvojen rajaaminen pois päivämäärän perusteella.

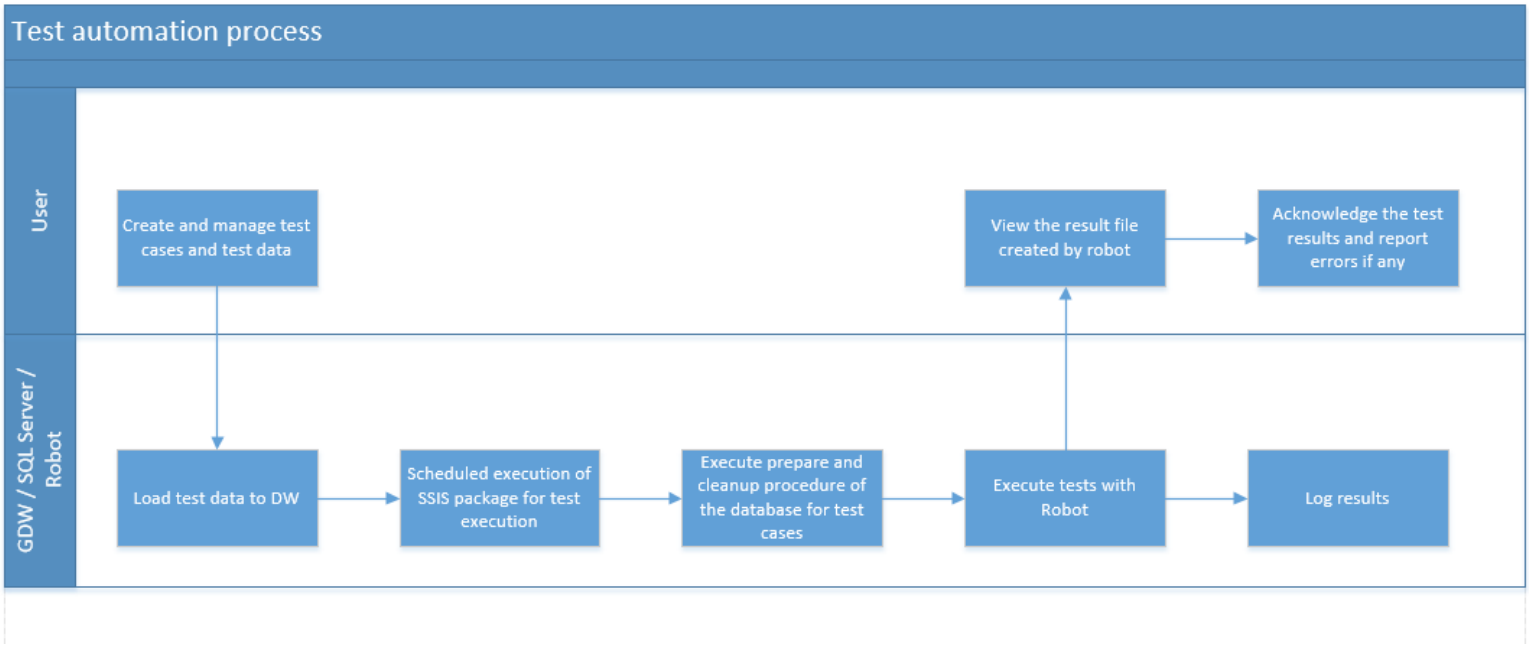
Suunnitelman alkuunsaamisen jälkeen aloin työstämään prototyyppiä komponentista, jotta näkisimme kuinka se käytännössä toimii. Tavoitteena oli havaita löytyykö siitä ja sen toiminnasta heti mahdollisia puutteita tai huonoja puolia. Prototyypin kehittämisen yhteydessä huomattiin melko nopeasti, että projektille annetuissa aikamääreissä komponentti ei tulisi täyttämään sille asetettuja vaatimuksia. Esimerkiksi erilaisten testitapauksen suorittamisesta ei olisi saatu riittävän kattavaa ja komponentin ylläpito olisi ollut liian työlästä. Tämän lisäksi virheiden todennäköisyys erilaisten käsittelysääntöjen toimeenpanossa olisi ollut todennäköistä. Ratkaisu olisi ollut kuitenkin toimiva, mutta aikataulun ja vaatimuksena olleen helppokäyttöisyyden vuoksi projektin onnistuminen olisi tällä ratkaisulla riskeerattu. Näiden huomioiden jälkeen projektitiimin päätös oli, että toteutustapa olisi ollut hyvä, mikäli aikaa olisi ollut käytettävissä enemmän. Projektin tiukan aikataulun vuoksi jouduttiin kuitenkin alkaa kartoittamaan toisia vaihtoehtoja.

Seuraavaksi projektitiimissä siirryttiin kartoittamaan muita vaihtoehtoja ja käytiin keskusteluja yrityksen muiden asiantuntijoiden kanssa, joilla on kokemusta datan käsittelyyn liittyvästä automaatiotestauksesta. Tarkoituksena oli käydä mahdollisimman monta erilaista vaihtoehtoa läpi mahdollisimman monesta erilaisesta näkökulmasta. Näissä keskusteluissa nousi esiin testiautomaatiotyökalu Robot Framework ja aloin tutkimaan sen sopivuutta käyttötarkoitukseemme.

Robot Frameworkin ollessa erittäin laajennettavissa oleva ja kustomoitava testiautomaatiotyökalu, päädyimme aloittamaan toisen prototyypin toteutusta Robot Frameworkilla. Toimintaa varten testaajan tai kehittäjän on luotava kahta erilaista testidataa taulukohtaisesti, joka pitää vaatimukset testaajille ajallisesti sekä teknisesti matalampana. Tiedon lähtötaululle staging-alueella on oltava oma Excel-tiedostonsa, jossa on sarakkeiden nimet kuten tietokannan taulussa. Tiedoston testidatan tulee kattaa mahdollisimman monta erilaista transformaatiota ja käsittelyä ETL-prosessista, jotta testaus on mahdollisimman kattavaa. Ladattavan testidatan lisäksi tietovarastoon päätyvästä tulosdatasta on oltava oma tiedostonsa CSV-muodossa. Tiedoston testidataa verrataan tietovarastoon päätyvää tietoa vastaan. Tässä vaiheessa testaajan ja

vaatimusmäärittelyiden merkitys nousee suureksi, koska jos ETL-prosessi ei vastaa sen määrittelyitä, odotetun tulosdatan rakentaminen on huomattavasti virhealttiimpaa.

Testaajan tekemä lähdedata ladataan normaalisti tietovarastoon ETL-prosessin kautta. Automaattitestaus suorittaa käyttäjän tekemien lähdedatojen ja odotetun tulosdatan vertailun automaattisesti testeissä kutsuttavan SQL-proseduurin avulla. Kun lataus on suoritettu, osana automaattitestausta siitä tehdään merkintä tietokantaan, jonka arvon Robot käy tarkastamassa. Mikäli tietovaraston testattavan taulun datassa ja odotetussa tulosdatassa ei ole eroja niin testin suoritus hyväksytään.



(Kuva 5. Testiautomaation prosessikaavio)

Kuvassa 5 on kuvattuna koko komponentin toimintalogiikka joka on kuvattuna tässä kappaleessa. Testiympäristössä suoritettava testiajo ajastetaan ja konfiguroidaan lataamaan data staging-alueelle testaajien ja kehittäjien luomista testidatatiedostoista, eikä oikeista tietolähteistä. Testiympäristöön on rakennettu myös SSIS-paketti, jonka avulla automaatiotestaus käynnistetään Robot frameworkin käynnistävällä komentorivitiedostolla. Kyseinen paketti suorittaa myös tarvittavat SQL-proseduurit, joiden avulla tietovarastossa historioidaan virheet ja tyhjennetään taulut uutta latausta varten. Tämä helpottaa regression havaitsemista, kun testit ajetaan joka yö ja testiraportit tarkastetaan päivittäin. Robot frameworkin testitapauksiin on määritelty käytettävät taulut ja tiedostot, jolloin niitä ei tarvitse kovakoodata itse SSIS-pakettiin. Tämän ansiosta komponentin käyttö on huomattavasti suoraviivaisempaa ja helpompaa, kun muutokset ja ylläpitotyö tehdään ainoastaan yhteen paikkaan. Tämä mahdollistaa myös komponentin käytön muissa Microsoftin työkaluja hyödyntävien projektien ETL-prosesseissa. Tämä oli

projektin lopputulokselta vaadittu tärkeä ominaisuus, koska yleiskäyttöisyys oli avaintekijä tämän komponentin hyödyntämisestä.

Robot frameworkin etuna tässä yhteydessä on myös testien tuloksen automaattinen raportointi, se luo testiajon jälkeen xml- ja html-tiedostot joissa on testien tulokset, joten testaajan on helppo tarkastaa ovatko kaikki testit suoritettu onnistuneesti. Tämä helpottaa eri kokonaisuuksien testien onnistumisen hahmottamista ja välitön palaute testeistä auttaa myös reagoimaan nopeammin testeissä ilmaantuviin virhetilanteisiin, sekä sen avulla voidaan havaita myös järjestelmässä ilmaantuvaa regressiota, kun kaikki testit suoritetaan tasaisin väliajoin.

Mikäli projektille varattua aikaa olisi ollut enemmän testauksen automatisoinnin käytössä, niin todennäköisesti ensimmäinen vaihtoehto, eli C#-pohjainen komponentti, olisi ollut valittu ratkaisu. Sen avulla testauksesta olisi saatu huomattavasti teknisempää ja tarkempaa. Robot Framework pystyy kuitenkin toteuttamaan vastaavat testitapaukset, eikä siten ollut kuitenkaan huono vaihtoehto. Sen etuina oli etenkin dynaamisuus ja testitapausten helppo luonti, mutta se ei kuitenkaan ole niin tekninen testityökalu kuin alusta asti tiettyyn käyttötarkoitukseen toteutettu komponentti. Robotin avulla toteutettu testaus mukailee white-box testausta.

4.3 Toteutus

Päätettyämme toteutustavan aloin toteuttamaan komponenttia testaamisen toiminnoista. Prioriteetiltään korkein tehtävä oli saada toimiva työkalu, joten mielestäni oli loogisinta aloittaa siitä, että komponentin toimintalogiikka ja toiminnallisuudet saadaan rakennettua ensin. Aluksi toteutin SQL-proseduurin, joka suorittaa automaattisesti vertailun tietovaraston taulun ja siihen liittyvän odotetun tulosdatan välillä. Proseduurin tehtävänä oli luoda väliaikaiset taulut testattavan latauksen tiedoille ja suorittaa vertailu tietovaraston ja tuodun CSV-tiedostossa olevan tulosdatan välillä. Proseduuri itsessään oli siis koko testauksen toteuttava logiikka, mutta ei sisältänyt automatisointiin liittyviä osuuksia. Proseduurikutsu rakennetiin toimimaan parametreillä, jotta proseduuria voidaan käyttää dynaamisesti datasta ja käytettävistä tauluista riippumatta. Proseduurin testaaminen yksinkertaisella datalla ja taulurakenteella onnistui helposti ja suurimmat ongelmat tässä vaiheessa liittyivät lähinnä komponentin toimintaympäristöön ja tiedostojen sisään lukuun. Edellä mainitut ongelmat liittyivät lähinnä teknologiaan ja olivat melko vaikeasti selvitettävissä, koska komponentin toimintatapa vaikutti olevan harvinaislaatuisesti toteutettu. Tämän seurauksena neuvojen ja ohjeiden erilaisista lähteistä oli vaikeaa, koska

tietoa ei löytynyt ja usein jouduin itse kartoittamaan ratkaisuvaihtoehtoja ongelmien ratkomiseksi.

Saatuani testien toimintalogiikan valmiiksi oli työlistalla seuraavana toteuttaa testien suorittaminen Robot Frameworkilla. Testien suorittamisen lisäksi täytyi toteuttaa niihin liittyvät ajastukset ja automatisointi. Robotin ollessa avoimen lähdekoodin työkalu ja siten kaikkien vapaasti laajennettavissa, oli siihen jo valmiiksi olemassa erilaisia kirjastoja. Käyttäjien tekemistä kirjastoista oli huomattavaa hyötyä projektiin liittyen. Yksi käyttämistämme kirjastoista, nimeltään Robot Framework database-library, osoittautui täydelliseksi projektin kannalta. Kirjastossa on toteutettuna ja valmiiksi käytettävissä erilaisia Robotin avainsanoja, joiden avulla voidaan suorittaa erilaisia SQL-komentoja halutuilla parametreilla. Loin robotille tarvittavat automaattitestit, jotka toimivat dynaamisesti riippumatta siitä mikä taulu tai tietokanta on kyseessä. Testidatan luomisen lisäksi ainoa asia mitä käyttäjän tarvitsee tehdä testin rakentamiseksi, on tehdä testin kutsuminen taulun ja sen odotetun tulosdatatiedoston nimillä. Kuvassa 6 on esitetty kuinka robotin resurssitiedosto, jota testitapauksissa käytetään, on luotu ja kuvassa 7 on esitetty kuinka käyttäjä toteuttaa testin suorittamisen testitapaustiedoston avulla. Kuvassa 7 testitapaukselle on annettu parametrit kuvan 6 avainsanan vaatimusten mukaisesti, jolloin kutsutaan oikeaa proseduuria oikealla testidatatiedostolla ja kohdennetaan vertailu oikeaan tauluun.

```
*** Settings ***
Library   Process
Library   String
Library   DatabaseLibrary

*** Variables ***
${DATABASE}   DW
${PROCNAME}   [ExecuteTest]
${FILEPATH}   [redacted]
${SERVERADDRESS}   [redacted]

*** Keywords ***

Validate ETL

   [Arguments]   ${FileName}   ${DataTable}
   Connect To Database Using Custom Params  pymssql database='${DATABASE}', user='[redacted]', password=[redacted], host='${SERVERADDRESS}', port=[redacted]
   Execute Sql String   USE ${DATABASE} EXEC ${PROCNAME} @Filepath = '${FILEPATH}', @Filename = '${FileName}', @DataTable = '${DataTable}'
   ${queryResult}   Query   USE ${DATABASE} SELECT FailedCount FROM TestResult WHERE TableName = '${DataTable}'
   Should Be Equal As Numbers   ${queryResult[0][0]}   0   ERROR: Rows in table do not match the given data!
   Disconnect From Database
```

(Kuva 6. Robotin käyttämä resource -tiedosto, josta poistettu arkaluontoiset tiedot.)

```
*** Settings ***
Resource   resource.robot
Suite Teardown   Terminate All Processes   kill=True


*** Test Cases ***
DW ETL FactSales
   [Tags]   ETL test
   Validate ETL     FactSales.csv   [FactSales_Test]
```

(Kuva 7. Robotin käyttämä testcase -tiedosto.)

Tämän vaiheen jälkeen suorittamisen jälkeen oivalsin, että testien suorituksesta on mahdollista historioida dataa tietokantaan ja päätimme toteuttaa testien historiointiin liittyvän ominaisuuden samalla kertaa. Testien historiointi tapahtuu siten, että ennen jokaista ajoa kutsutaan SQL-proseduuria, joka siirtää edellisen testiajon tulokset omaan tauluunsa tietovarastossa ja yksilöi jokaisen ajokerran omalla ID-luvulla. Tämä tieto on mielenkiintoista ja hyödyllistä, kun halutaan seurata ajohistoriaa ja tehdä havaintoja esimerkiksi siitä missä taulussa havaitaan useimmin virheitä testiajoissa.

Muutaman yksinkertaisen testin onnistuttua ja komponentin osien toimivuuden varmistuttua päätimme siirtää komponentin kehityspalvelimelle. Kehityspalvelimella toteutetaan yhtä toimeksiantajayrityksen projekteista. Testiautomaatiokomponentin toteutus liittyi myös kyseiseen projektiin. Projektissa ei vielä toteutushetkellä ollut ETL-prosesseja toteutettuna ja koimme myös tärkeäksi saada komponentin käyttöön heti alusta asti. Käyttöönotto alkuvaiheessa on tärkeää, koska silloin sen käytöstä muodostuu toimintatapa eikä sen käyttöönotto toteutuksen alkamisen yhteydessä vaadi kovinkaan paljon ylimääräistä ponnistelua tai testidatan luomista takautuvasti.

Siirto aloitettiin asentamalla kehityspalvelimelle Python, jonka avulla Robot framework toimii, ja siihen liittyvät muut projektin toteutuksessa käytetyt kirjastot ja tarpeelliset ominaisuudet. Tämän jälkeen aloin konfiguroimaan kehitysympäristöä testien suorittamista varten. Robot frameworkin toimintakuntoon saaminen oli konfiguroinnin yksinkertaisin vaihe, koska siihen liittyen täytyi määritellä uudelleen ainoastaan tietokantapalvelimen osoite, nimi ja testitiedostojen hakemistot. Suurimmat ongelmat tulivat tälläkin kertaa silloin, kun testien suorittamista alettiin toteuttamaan. Testien ajaminen manuaalisesti käynnistämällä Robottia kutsuva komentorivitiedosto onnistui mutkattomasti, mutta SQL-serverin omalla ajastuksella käynnistettynä testit eivät toimineet. Tämän seurauksena tulleiden virheilmoitusten läpikäyminen oli haastavaa niiden epäkuvaavien virhekoodien takia ja selvitystyö vei tässäkin vaiheessa suurimman osan ajasta. Aiemmin prototyyppiä tehdessä olin kuitenkin saanut samankaltaisia virheilmoituksia. Olin jo silloin onnistunut kohdentamaan ongelman syyn ja tälläkin kertaa

kyseessä oli puutteelliset käyttöoikeudet. Oletusarvoisesti automaattitestausta suorittava käyttäjätunnus ei voinut kutsua tietokantapalvelimella komentorivitiedostoa, joten sen kutsu siirrettiin osaksi SSIS-pakettia. Muutoksen jälkeen komentorivitiedoston kutsu ja odotetun tulosdatan sisään luku tietokantaan saatiin lopulta toteutettua.

Tässä vaiheessa toteutettiin myös testitulosten historioinnin automatisointi. Tiedot historioidaan omaan tauluunsa testiajokohtaisesti ja ne on yksilöity testiajon mukaan. Tämä toiminto toteutettiin SQL-proseduurilla, joka siirtää nykyisen testiajon tiedot historiatauluun ja tyhjentää tämän jälkeen nykyisen ajon tiedot. Kun tiedot tallennetaan samaan tauluun yksilöitynä testiajon ja testitapauksen mukaan, on virheiden seuranta helpompaa. Tämän lisäksi tieto voidaan viedä raportille ja seurata esimerkiksi epäonnistuneiden testitapausten määrää suhteessa aikaan. Mikäli virheitä ilmentyy paljon, niin raportin kautta on mahdollista havaita ongelmakohdat ja siten niihin voidaan puuttua ajoissa.

Toteutusvaiheen yhteydessä syntynyt dokumentaatio kirjoitettiin käyttöohjemuodossa, koska dokumentti on ainoastaan yrityksen sisäiseen käyttöön. Tämän lisäksi ympäristön konfiguroinnista tehtiin oma erillinen dokumenttinsa, jolloin komponentin käyttöönotto ja käyttö muissa projekteissa on helpompaa ja suoraviivaisempaa. Ohjeet sisälsivät myös kuvauksen vaaditusta testidatasta sekä siihen liittyvästä odotetusta tulosdatasta.

4.4 Testaus

Tietovarastoprojekti, jonka osana komponenttia toteutettiin, ei ollut vielä riittävän pitkällä kehitysvaiheessa, jotta testiautomaatiokomponenttia olisi voinut testata osana ETL-ajoja. Komponentin testaus kuitenkin toteutettiin luomalla sitä varten oma ympäristönsä. Ympäristöön luotiin oma tietokanta tietovarastolle, sekä testidataa joka pohjautui tietovaraston tauluihin. Tarkoituksena tässä vaiheessa oli saada mahdollisimman realistinen ja mahdollisen asiakkaan ympäristöä peilaava testiympäristö, jossa automaattitestit suoritetaan ajastetusti ja niiden tulokset raportoidaan, kuten käyttöönoton jälkeen on tarkoitus.

product_code	amount	price_unit	timestamp	employee_code	store_code
aa15	4	5,5	2016-11-03 12:03:05	emp_10042	Store_1
cde222	2	3,45	2016-11-03 12:03:05	emp_10042	Store_1
cbk4	20	0,75	2016-11-03 14:53:13	emp_1999321	Store_1
aa15	4	5	2016-11-03 17:11:44	emp_174582	Store_1
kf11	3	3,25	2016-11-03 17:11:44	emp_174582	Store_1
acl-104	2	1,1	2016-11-03 17:11:44	emp_174582	Store_1

(Kuva 6. Esimerkki testauksessa käytettävästä lähdedatasta)

sales_id	product_id	amount	price_unit	price_total	timestamp	employee_id	store_id
1	14	4	5,5	22	2016-11-03 12:03:05	7	1
1	3	2	3,45	6,9	2016-11-03 12:03:05	7	1
2	113	20	0,75	15	2016-11-03 14:53:13	3	1
3	14	4	5	20	2016-11-03 17:11:44	14	1
3	25	3	3,25	9,75	2016-11-03 17:11:44	14	1
3	53	2	1,1	2,2	2016-11-03 17:11:44	14	1

(Kuva 7. Esimerkki odotetusta tulosdatasta.)

Testausta varten toteutettiin yksinkertainen ETL-ajo, jonka avulla automaattitestien lähtödatasta saatiin dataa tietokantatauluun. Tämän jälkeen Robot Framework ohjelmoitiin toimimaan kyseisten testidatatiedostojen, jotka on esitetty kuvissa 6 ja 7, sekä testiympäristön tietovarastoon luotujen taulujen kanssa. Testidatatiedostot tallennettiin palvelimelle helpompaa ylläpitoa ja käyttöä varten. Sen lisäksi luotiin erillinen SSIS-paketti, jonka tarkoituksena oli suorittaa kaikki Robotille luodut testitapaukset. Kyseinen paketti ajastettiin palvelimelle ajettavaksi kerran vuorokaudessa ja sen avulla varmistuttiin ajastuksen oikeasta toiminnasta.

Testaukselle ei projektin tiukan aikataulun vuoksi saatu varattua montaa päivää aikaa, mutta kehityksen yhteydessä tapahtuneen testauksen turvin voitiin olla varmoja, että komponentti on toimiva. Testausvaiheelta aikaa söi suunnittelu ja prototyyppien valmistelu, koska tämän kaltainen komponentti oli uusi sekä toimeksiantajalle, että toteuttajille.

5 Ajatukset projektin jälkeen

Opinnäytetyöprojektin tarkoituksena oli suunnitella ja toteuttaa yleiskäyttöinen testiautomaatiokomponentti ETL-prosessin validointia varten. Projektilla oli määrällisesti vähän aikaa, eikä tuloksena odotettu kaiken kattavaa testiautomaatiotyökalua vaan ikään kuin runkoa tulevaisuudessa laajennettavalle ja paranneltavalle komponentille. Tämän lisäksi sen tuli toimia datasta ja tietokannasta riippumatta kaikissa Microsoft-pohjaisissa tietovarastolatauksissa.

Oma mielenkiintoni testiautomaatioon ja ylipäätään testaamiseen on opiskelujeni ja työurani aikana ollut aina korkea, koska lähes poikkeuksetta monissa IT-projekteissa törmätään erilaisiin puutteisiin laadunvarmistuksessa. Etenkin kattava testaus ja regressiotestaus ovat asioita, joiden avulla voidaan huomata virheet aiemmin toteutetuissa toiminnallisuuksissa ja ominaisuuksissa. Testien suorittaminen vie kuitenkin aikaa ja vaivaa mikäli se tehdään manuaalisesti, jolloin kokonaisvaltainen testaaminen jää usein tekemättä, mikäli automatisoinnin mahdollistavia työkaluja ei ole käytössä. Tämän vuoksi automaattitestauksen kasvava suosio nykypäivänä ei ole mielestäni yllätys, koska se mahdollistaa resurssien vapauttamisen haastavampiin testaustöihin, joita ei kannata tai voi automatisoida.

Projektin toteutuksessa huomioin parhaalla mahdollisella tavalla sen, että komponentin jatkokehitystä tullaan tekemään jatkossa varmasti. Tällaisia huomioita tein etenkin parametrisointiin ja toiminnallisuuksien riippumattomuuteen liittyen ja sen takia tein komponentin toteutuksen niin, ettei se ole ympäristösidonnainen. Komponentista tuli dynaaminen niin, ettei se ole sidonnainen käytettävään tietokantaan, dataan, tauluihin tai oikeastaan mihinkään muuhun kuin Microsoft SQL server-teknologiaan, joka oli itselleni tärkein onnistuminen projektissa. Mahdollisia jatkokehitystarpeita huomasin itse jo projektin aikana ja valtaosa niistä liittyy komponentin käytettävyyteen ja tulosten seuraamiseen. Robot frameworkin testiajon jälkeen tuottamat lokiraportit olisi mielestäni hyvä siirtää oman hakemistonsa alle ennen jokaista testiajoa, jonka voi toteuttaa esimerkiksi samankaltaisella komentorivitiedostolla kuin automaattitestien käynnistämisen. Tämän ansiosta voisi siten myös yksittäisten ajojen tietoja tarkemmin ja tallentaa tarkat tiedot testiajosta. Kun testiajot on historioitu ja niiden tiedot ovat XML-tiedostoformaatissa, voidaan niitä esimerkiksi analysoida ja vertailla keskenään mikäli se koetaan tarpeelliseksi. Kuitenkin tietovarastoon tallennettavat tiedot testien suorittamisesta ovat tällä hetkellä mielestäni enemmän kuin riittävät ajojen suorituksen seurantaan.

Komponentin tarkoituksena on toimia erilaisissa Microsoft-pohjaisissa tietovarastoissa, joten mielestäni ennen jatkokehitystä komponentti ja sen käyttämiseen vaadittavat työkalut, kirjastot ja ohjeet on hyvä tallentaa yhtenä pakettina. Tämä nopeuttaa työkalun käyttöönottoa uusissa ympäristöissä ja tekee siitä myös suoraviivaisempaa, eikä esimerkiksi työkalujen, ja etenkään Pythonin versioiden, kanssa synny yhteensopivuusongelmia. Nopea käyttöönotto ja testiautomaation mukaan ottaminen projektin alkuvaiheessa luovat hyvät käytännöt projektille testauksen osalta, eikä mahdollisia virheitä pääse livahtamaan mukaan kehitysvaiheen aikana. Mitä aiemmassa vaiheessa projektia automaattitestaus otetaan mukaan osaksi testausta, sitä suuremmat hyödyt siitä kertyy pitkässä juoksussa.

Omasta mielestäni opin projektin aikana todella paljon uutta erilaisilta aihealueilta, koska pääsin osallistumaan kehitysprojektin eri vaiheisiin vastuullisessa roolissa ja minulla oli vapaat kädet toteuttaa komponentti parhaaksi katsomallani tavalla. Tämä pakotti punnitsemaan asioita eri näkökulmista ja tekemään analyyttista vertailua erilaisten vaihtoehtojen välillä. Mielestäni tämä on välttämätön taito asiantuntijatyössä ja sen merkitystä ei voi väheksyä. Tämän lisäksi etenkin Robot Framework oli itselleni käytännön tasolla vieras työkalu ja opin projektin aikana siitä ja sen käyttötarkoituksista merkittävästi. Tämän takia uskon, että pystyn jatkossakin työskentelemään sen parissa helposti ja soveltaa projektin aikana oppimiani asioita myös muissa tapauksissa. Myöskin erilaiset määrittely- ja suunnittelutehtävät olivat hyödyllisiä oman oppimisen kannalta, koska se vahvisti omaa ymmärrystäni siitä, että toteutuksen kuvaus on tehtävä mahdollisimman tarkasti ja yksinkertaisesti. Tämä johtuu siitä, että ihmiset voivat usein ymmärtää saman asian eri tavoin riippuen siitä kuinka se selitetty auki ja kuvattu ja myös siitä. Ja myös siitä minkälainen kokemus ja aiempi osaamistaso asiaan liittyen henkilöillä on.

Mielestäni projektia oli mukava tehdä yksin, koska vastuu ja mahdollisuus vaikuttaa omaan tekemiseen ja ratkaisutapoihin oli ammatillisesti kasvattavaa sekä omaa osaamistasoani kehittävää. Yleensä kuitenkin pidän henkilökohtaisesti enemmän siitä, että saan keskustella muiden kehittäjien kanssa eri toteutustavoista ja ongelmista, joka yleensä antaa uutta näkökulmaa kehitystyöhön. Oman mielipiteen ja ajatuksen perustelu pakottaa huomioimaan myös mahdollisia heikkoja puolia ratkaisussa ja tällä tavalla usein erilaisten ratkaisujen heikot puolet tulevat nopeammin esille, kuten esimerkiksi ensimmäinen prototyyppi, jota projektissa työstettiin. Tähän liittyen kuitenkin minulla oli onnekseni tarvitessani tukena yrityksen muita asiantuntijoita, joilla oli osaamista testiautomaatioon ja Robot Frameworkiin liittyen, jolloin pääsin tuomaan työn aikana ilmaantuneita ongelmia esiin ja käymään keskustelua erilaisista ratkaisuvaihtoehdoista niihin liittyen.

Loppujen lopuksi päällimmäisenä asiana kuitenkin mieleen jäi ajanhallinta, koska projektille varatut resurssit olivat ajallisesti pieniä ja tekemiseen ei loppujen lopuksi jäänyt aikaa kuin noin 15 henkilötyöpäivää. IT-projekteissa pitää mielestäni kyetä huomaamaan tilanteet, joissa projekti ei etene ja tehdään selvityksiä. Tämän kaltaisissa tilanteissa pitää pystyä tekemään päätöksiä vaikka olisikin puutteellista tietoa, koska muuten projekti jää varmasti aikataulustaan.

Lähteet

Atre, S. & Moss, L. 2004. Business Intelligence Roadmap: The Complete Project Lifecycle for Decision-Support Applications. 3. painos. Yhdysvallat.

Doan, A., Halevy & A. Ives, Z. 2012. Principles of Data Integration. E-kirja.

Dustin, E., Rashka, J. & Paul, J. 2008. Automated Software Testing: Introduction, Management and Performance. 13. painos. Yhdysvallat.

Kimball. R., Reeves, R., Ross, M. & Thorntwaite, W. The Data Warehouse Lifecycle Toolkit. E-kirja. Wiley. Yhdysvallat.

Levy, E. & Oates, J. 2007. What is the difference between data warehouse testing and conventional testing?. Luettavissa: http://www.information-management.com/news/ask_the_experts/-1080612-1.html Luettu: 3.11.2016.

Paredes, J. 2013. The Multidimensional Data Modeling Toolkit: Making Your Business Intelligence Applications Smart with Oracle OLAP. 1. painos. Olap Word Press. Yhdysvallat.

Ponniah, P. 2001. Data Warehousing Fundamentals: A Comprehensive Guide for IT Professionals. 1. painos. Wiley. Yhdysvallat.

Rainardi, V. 2007. Building a Data Warehouse: With Examples in SQL Server. 1. painos. Apress. Yhdysvallat.

Räsänen, S. 2013. Ohjelmiston testaus ja laatu. Luettavissa: http://webd.savonia.fi/home/ktrasse/muut/testaus_laatu/testaus_2.pdf Luettu: 26.11.2016.

Serra, J. 2013. Why You Need A Data Warehouse. Luettavissa: http://www.jamesserra.com/archive/2013/07/why-you-need-a-data-warehouse/ Luettu: 14.11.2016.

Silvers, F. 2008. Building and Maintaining a Data Warehouse. 1. painos. Auerbach Publications. Yhdysvallat.

Tortorella, N. 2015. Kuva tietovarastoinnin ETL-prosessista. http://www.neiltortorella.com/data-warehouse-definition/inspiring-data-warehouse-definition-4-data-warehouse-data-warehousing/ Luettu: 23.10.2016.

Vercellis, C. 2011. Business Intelligence: Data Mining and Optimization for Decision Making. E-kirja. John Wiley & Sons. Iso-Britannia.