jamk.fi

Tuuliturbiinivaihteen kokoonpano- ja komponenttiraportoinnin automatisointi

Tuomas Kiiski

Opinnäytetyö
Joulukuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), automaatiotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Kiiski, Tuomas	Opinnäytetyö, AMK	Joulukuu 2018
	Sivumäärä 45	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Tuuliturbiinivaihteen kokoonpano- ja komponenttiraportoinnin automatisointi

Tutkinto-ohjelma
Automaatiotekniikan tutkinto-ohjelma

Työn ohjaaja(t)
Kuisma, Ari

Toimeksiantaja(t)
Moventas Gears Oy, Jukka Paananen

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda järjestelmä tuuliturbiinivaihteen kokoonpanoprosessin ja jäljitystietojen raportointiin. Järjestelmällä piti pystyä luomaan kokoonpano-ohjeita, määrittämään työvaihekohtaisesti kerättävät jäljitystiedot, komponenttien tyypit, mittaukset toleransseineen sekä luomaan tarkistuslistat.

Opinnäytetyön toimeksiantaja oli tuuliturbiinivaihteita valmistava Moventas Gears Oy. Yrityksellä on tuotantoa Jyväskylän ja Karkkilan toimipisteissä.

Järjestelmä kirjoitettiin C# kielellä hyödyntäen WinForms-luokkakirjastoa käyttöliittymän luomiseen. Tietokanta toteutettiin MySQL-tietokantajärjestelmällä. Järjestelmä koostui tietokannasta, yhteisestä luokkakirjastosta sekä näitä käyttävistä Assembly Manager ja Assembly Designer sovelluksista.

Teoriaosuudessa käydään läpi perusteet tietokannoista ja sen relaatioista, C#-kielestä, sen ominaisuuksista ja paradigmoista sekä olio-ohjelmoinnin perusteista. Valmistunut järjestelmä perustuu teoriaosuudessa käytyihin asioihin. Toteutusosuudessa käydään läpi valmiiden sovellusten ominaisuuksien ja käyttämisen lisäksi tuuliturbiinivaihteen kokoonpanoprosessi.

Valmiista työstä saatiin toimeksiantajan toiveiden mukainen järjestelmä kokoonpanoprosessin ohjeiden luontiin ja jäljitystietojen keräykseen. Kehitystyön aikana lisättiin vielä käytetyn työkalun kirjaus osaksi järjestelmää. Järjestelmä huolehtii tallennetun data eheydestä ja tarjoaa työkalut sen hallinnointiin. Assembly Reporter -sovellus on suunniteltu huolehtimaan syötetyn tiedon oikeellisuudesta.

Avainsanat (asiasanat)
C#-kieli, MySQL, WinForms, kokoonpanosuunnitelma, jäljitettävyys, olio-ohjelmointi

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Kiiski, Tuomas	Bachelor's thesis	December 2018
		Language of publication: finnish
	Number of pages 45	Permission for web publication: x

Title of publication

Report automation of assembly process and components of wind turbine gearbox

Degree programme

Automation Engineering

Supervisor(s)

Kuisma, Ari

Assigned by

Moventas Gears Oy, Paananen Jukka

Abstract

The goal of the thesis was to create a system for documenting the assembly process and traceable components of wind turbine gear. The system was to be able to create assembly instructions, define traceable components by their workphase, types of components, measurements with their specification limits and create checklists.

The thesis was assigned by Moventas Gears Oy, which manufactures wind turbine gearboxes in its Jyväskylä and Karkkila plants.

The system was written in C# language making use of WinForms class library for creating graphical user interface. The database was implemented with MySQL database management system. The system constitutes of a database, shared class library, and two applications using them, namely Assembly Manager and Assembly Reporter.

In the theoretical part, the basics of the database and its relations, C# language and its programming paradigms, as well as object-oriented programming are reviewed. The completed system rests on these theories. In the implementation section, the features and the use of the produced applications are reviewed, as well as the manufacturing process of wind turbine gearboxes.

The completed system meets the assigners' initial wishes and needs. During the development, documentation of the used tool was added as part of the system. The system keeps the data valid and offers tools for managing it. Assembly Reporter application is designed to accept only valid inputs.

Keywords/tags (subjects)

C#-language, MySQL, WinForms, manufacturing process plan, traceability, object-oriented programming

Miscellaneous

Sisältö

Kuviot

Taulukot

1 Johdanto

1.1 Tavoitteet

Opinnäytetyön tavoitteena oli luoda tuuliturbiinivaihteen kokoonpano- ja komponenttiraportointia helpottava ja automatisoiva tietojärjestelmä. Järjestelmä mahdollistaa joustavan kokoonpano-ohjeistuksen luomisen, tiedon keräämisen sekä kokoonpanoprosessin dokumentoinnin.

Työ koostuu kolmesta osasta:

- AssemblyManager, sovellus kokoonpano-ohjeiden määrittämiseen,
- AssemblyReporter, kokoonpano-ohjeiden näyttäminen ja tietojen keruu
- Tietokanta tallentamiseen.

AssemblyManagerin ominaisuudet:

- Vaihdemallille kohdennettujen alikokoonpanojen luonti
- Alikokoonpanon jakaminen työvaiheisiin
- Työvaihteelle pitää pystyä määrittämään ohje (vapaa teksti), kerättävät jäljitystiedot, kirjattavat mittatulokset sekä niiden toleranssit, käytettävän työkalun tyyppi sekä kuva
- Päivitettyjen ohjeiden pitää näkyä tuotannon puolella heti.

AssemblyReporterin ominaisuudet:

- Uusimman kokoonpano-ohjeen näyttäminen
- Määriteltyjen tietojen kerääminen
- Mahdollisuus välitallennukseen
- Yleisnäkymä vaihteeseen asennettavista komponenteista sekä tärkeistä mitoista

1.2 Toimeksiantaja

Työn toimeksiantaja on Jyväskylässä tuuliturbiinivaihteita valmistava Moventas Gears Oy. Tuotanto tapahtuu kahdessa Jyväskylän toimipisteessä, mutta yrityksellä on 8 huolto- ja myyntipistettä Euroopassa, Pohjois-Amerikassa sekä Aasiassa. Moventas työllistää globaalisti yli 500 ihmistä. Yrityksen edeltäjä Valmet on valmistanut teollisuusvaihteita 1940-luvulta lähtien, josta tuuliturbiinivaihteita valmistava Moventas irtautui yli 35 vuotta sitten. (Moventas Gears Oy n.d.)

1.3 Lähtökohdat ja tutkimusmenetelmät

Ennen opinnäytetyötä, kokoonpano-ohjeet olivat yksittäisiä dokumentteja. Komponenttien jäljitystiedot kerättiin erillisiin Excel-tiedostoihin. Opinnäytetyön tarkoituksena oli yhdistää ohjeistus ja tietojen kerääminen samaan järjestelmään. Tällä tavoin helpottuisi koko prosessin ylläpito. Myös tietojen hyödyntäminen jälkikäteen olisi mutkatonta. Järjestelmällä pystytään keräämään myös yksityiskohtaisempaa tietoa, kuten kokoonpanoprosessin läpimenoaikoja työpisteittäin.

Tutkimusmenetelmä perustui tulevien käyttäjien haastatteluun. Käyttöliittymän eri ratkaisuista käytiin dialogia, jonka perusteella lopullinen tulos muotoutui. Teknisessä toteutuksessa tukeuduttiin ongelma kerrallaan verkon lähdemateriaaleihin.

1.4 Sanasto

Seuraavassa taulukossa on lueteltu raportissa käytettyjä termejä.

Taulukko 1. Sanasto

Termi	Selitys
attribuutti	Luokan yksittäinen nimetty arvo
boolean	Tietotyyppi, jonka arvo voi olla joko ’true’ tai ’false’.
funktio	Kutsuttava aliohjelma, joka antaa paluuarvon
ilmentymä	Luokasta alustettu olio
lambda-kalkyyli	Ohjelmointikielen matemaattinen malli
luokka	Staattinen rakenne, joka määrittää miten olio käyttäytyy
metodi	Kutsuttava aliohjelma
olio	Loogisesti yhteen kuuluvat arvot ja metodit.
parametri	Metodille tai funktiolle annettava alkuarvo sitä kutsuttaessa
redundanssi	Toistavuus, esimerkiksi sama tieto tallennettu kahteen eri paikkaan
tietue	Tietokantarivi

2 Tietokanta

Kirjatut tiedot sekä kokoonpano-ohjeet tallennetaan yhteiseen tietokantaan. Tietokanta on tiedon tallentamiseen ja sen tehokkaaseen hakuun käytettävä järjestelmä, josta saadaan helposti yksittäisiä tietoja tai tietoyhdistelmiä kyselyillä (query). Yleisimmät tietokannat perustuvat joko relaatio- tai oliomalliin (Machajewski n.d.). Tässä työssä käytettävä MySQL-kanta perustuu relaatiomalliin.

2.1 MySQL

Tässä työssä käytetty MySQL on avoimen lähdekoodin relaatiotietokantajärjestelmä. Suomalainen Michael ”Monty” Widenius ja ruotsalainen David Axmark kehittivät järjestelmän 1995, ja sen ensimmäinen versio julkaistiin seuraavan vuonna. Muihin tietokantajärjestelmiin verrattuna MySQL tarjoaa verrattain paljon työkaluja (ilmaiseksi), ja siinä on rajapinnat moneen muihin kieliin, kuten C# ja Visual Basic, Java (JDBC) sekä ASP (ODBC). Myös tuki HTSQL:lle tulee mukana. MySQL:n omistaa Oracle Corporation. (PHP MySQL Database n.d.)

2.2 SQL-kieli

SQL (Structured Query Language) on kieli, jolla tietokantaa käytetään. Tietokannan asetukset, taulujen luonti sekä tiedon manipulointi ja haku tapahtuvat SQL-kielellä. MySQL-tietokantajärjestelmä käyttää omaa SQL-kieleen perustuvaa murrettaan (kuten myös muut relaatiotietokantajärjestelmät). (Introduction to SQL n.d.)

SQL on deklaratiivinen kieli, eli ratkaisu saadaan algoritmin kirjoittamisen sijaan esittämällä haluttu lopputulos. Deklaratiivinen ohjelmointi on imperatiivisen ohjelmoinnin vastakohta. (Mundy 2017.)

Vaikka olemassa on graafisia käyttöliittymiä MySQL:n hallintaan, tämän raportin esimerkeissä keskitytään SQL-kieleen.

2.3 Taulut, rivit ja sarakkeet

Tietokanta koostuu tauluista. Tauluille on määritelty tietyt sarakkeet. Jokaisella sarakkella on jokin tietotyyppi, joka kertoo millaista tietoa sarakkeelle voi tallentaa. Esimerkiksi int-tyyppiselle sarakkeelle voidaan tallentaa vain kokonaislukuja. (SQL Syntax n.d.)

Tietokantaan tallennetaan yleensä tosimaailmaa kuvaavia ilmentymiä riveiksi. Esimerkiksi taulussa ’komponentit’ voi olla sarakkeet ’piirustusnumero’ ja ’sarjanumero’, ja rivillä tiedot ’GDRM101010’ ja ’171354’ (Kuvio 1). Tässä taulussa jokainen rivi kuvastaa yhtä tosimaailman komponenttia (ilmentymää).

id	piirustusnumero	sarjanumero
1	GDRM101010	171354
2	GDRM101010	171355
3	GDRM101328	171026

Kuvio 1. Komponentit-taulu

2.4 Primary Key

Id-sarake (Kuvio 1) on erityinen Primary Key-sarake. Taulussa voi olla määriteltynä vain yksi Primary Key, mutta tämä ei ole pakollista. Primary key on uniikki, identifioiva kenttä, joka erottaa ilmentymät toisistaan. Tämä on yleensä automaattisesti lisääntyvä kokonaisluku, mutta myös tekstiä voidaan käyttää Primary Key:nä. Sosiaaliturvatunnus voisi olla esimerkki tällaisesta avaimesta. Tietueen lisääminen, jolla ei ole uniikkia Primary Key:tä, generoi errorin. (SQL – Primary Key n.d.)

2.5 Taulun luonti

Tietokantataulun ominaisuudet määritellään skriptissä, joka luo taulun (Kuvio 2). Rivillä 3 käsketään luomaan ”komponentti”-niminen tietokantataulu. Sulkeiden sisään lisätään sarakkeiden nimet sekä tietotyypit. Skriptissä käytetyt tietotyypit ovat:

- INT, integer, kokonaisluku. Juokseva kokonaisluku on perinteinen Primary Key
- VARCHAR, variable-length string of characters. Teksti, jonka maksimipituus on määritetty. Koska piirustusnumerossa on aina 10 merkkiä, sarakkeen tietotyypiksi on hyvä valita VARCHAR(10). Tällöin tiedon hakeminen nopeutuu, ja säästetään levytilaa. Rivejä voi olla miljoonia, ja tällöin turha muistin varaus hidastaa hakua turhaan. Yli 10 merkin pituisen tekstin tallentaminen generoi errorin.
- TEXT, vapaamittainen teksti. Sarjanumerolle ei ole maksimipituutta, joten tämä tietotyyppi on turvallisin valinta.

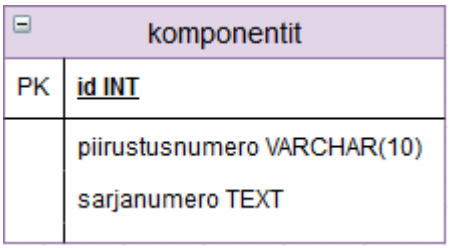
```
1   # risuaita merkkaa kommenttia
2   -- myös viiva-viiva-välilyöntiä voidaan käyttää
3   CREATE TABLE komponentti(          # luo taulu nimeltä komponentti
4                                      # määritellään taulun sarakkeet ja tietotyypit
5       id INT AUTO_INCREMENT,         # Integer-tyyppinen id-sarake,
6                                      # auto_increment huolehtii juoksevasta numeroinnista
7       piirustusnumero VARCHAR(10),   # Piirustusnumero, max. 10 merkkiä pitkä
8       sarjanumero TEXT,              # Sarjanumero, vapaata tekstiä
9       PRIMARY KEY (id));             # ID-sarakkeesta Primary Key
```

Kuvio 2. Komponentti-taulun luominen

Rivillä 9 kerrotaan että ID-sarakkeesta tulee Primary Key. Käsky loppuu puolipilkkuun. Sisennyksillä ja rivinvaihdoilla ei ole erityistä merkitystä, mutta niiden avulla skriptistä saa helpommin luettavamman.

2.6 Entity-Relation -kaavio

ER-kaaviota käytetään tietokannan suunnitteluun sekä visuaaliseen mallintamiseen. Kaavioon merkataan taulu ja sen sarakkeet, mutta siinä ei esitetä tietokantaan tallennettuja tietueita (Kuvio 33).



Kuvio 3. Komponentti-taulun ER-kaavio

Taulun yläosaan kirjoitetaan taulun nimi. Sarakkeet kirjoitetaan omille riveilleen. Sarakkeen nimen perään merkitään tietotyyppi. PK tarkoittaa, että id-sarake on Primary Key. Sarakkeen nimi ja tietotyyppi lihavoidaan ja alleviivataan. (What is ERD? n.d.)

2.7 Relaatiotietokanta

Relaatiotietokannassa sarake voi viitata toiseen tauluun. Tämä mahdollistaa tärkeitä ominaisuuksia:

- Hierarkkisen tiedon tallentaminen
- Vähentää saman tiedon tallentamista useaan kertaan (redundanssi)
- Lisää tiedon eheyttä ja sen luotettavuutta
- Hakeminen ja tiedon päivitys nopeutuvat

Sarakkeen viittaus toiseen tauluun tapahtuu Foreign Key -rajoitteella. Tämä tarkoittaa sitä, että tauluun lisätään sarake, johon tallennetaan vain jokin viitattavasta taulusta löytyvä Primary Key. Rajoite tehdään samalla kun luodaan taulu (mutta voidaan lisätä myös jälkeenpäin). Viittaavan sarakkeen tietotyypin tulee olla sama kuin viitattavan taulun Primary Key. Tällä tavoin pystytään tallentamaan hierarkkista tietoa, jossa yhdelle ilmentymälle kuuluu nolla tai useampi toisen tyyppistä ilmentymää. Esimerkiksi yhdellä asiakkaalle voi kuulua nolla tai useampi tilaus. (Rouse & Biscobing n.d.)

2.7.1 Tietokannan normalisointi

Tietokannan normalisointi on menetelmä, jolla tietokannan rakenne muutetaan sellaiseen muotoon, jossa redundanssisuus vähenee mahdollisimman paljon (Normalization of Database n.d.). Otetaan esimerkiksi komponentit-taulu (Kuvio 1). Tietueilla

1 ja 2 on sama piirustusnumero, eli yhdellä piirustusnumerolla voi olla monta sarjanumeroa. Ilman normalisointia piirustusnumero tallennettaisiin niin monta kertaa kuin sarjanumerokin.

Eritellään komponentit (piirustusnumero) ja sarjanumerot eri tauluihin. Esimerkin vuoksi lisätään piirustusnumeroon komponentin nimi ja sarjanumerolle kirjauspäivämäärä (Kuvio 4).

```
1  CREATE TABLE komponentti(                    # Luodaan komponentti-taulu
2      id INT PRIMARY KEY AUTO_INCREMENT,       # Primary key voidaan määritellä myös näin
3      nimitys TEXT,
4      piirustusnumero VARCHAR(10));
5
6  CREATE TABLE sarjanumero(
7      id INT PRIMARY KEY AUTO_INCREMENT,
8      numero TEXT,                             # Sarake ei voi olla samanniminen kuin taulu
9      pvm DATE,
10     komp_id INT,
11     # FK-rajoite, sarake komp_id viittaa komponentti-taulun id-sarakkeeseen
12     FOREIGN KEY (komp_id) REFERENCES komponentti(id));
```

Kuvio 4. FK-rajoitteen luonti

Ensimmäisenä luodaan komponentti-taulu. Primary Key:n voi määrittää samalla rivillä kuin sarake luodaan (rivit 1 ja 7). Tämän jälkeen voidaan luoda sarjanumero-taulu, joka viittaa komponentti-tauluun. Foreign Key -rajoitetta ei voi luoda ennen kuin viitattava taulu on luotu. Rivillä 12 luodaan rajoite. Tälle riville kirjoitetaan viittaava sarake sekä viitattava taulu ja sen sarake.

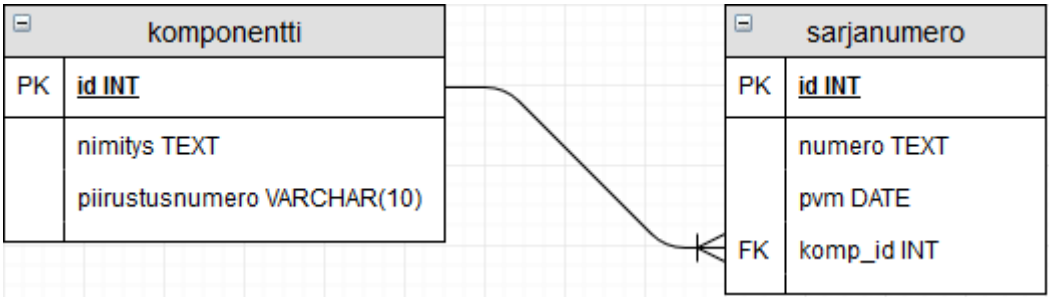
komponentti

id	nimitys	piirustusnumero
1	Iso planeetankantaja	GDRM101010
2	Pieni planeetankantaja	GDRM101328

sarjanumero

id	numero	pvm	komp_id
1	171354	1.6.2018	1
2	171355	2.6.2018	1
3	171026	3.6.2018	2

Kuvio 5. Normalisoitu tietokanta. Sarjanumero-taulussa komp_id -sarake kohdentaa tietueen tietylle komponentille



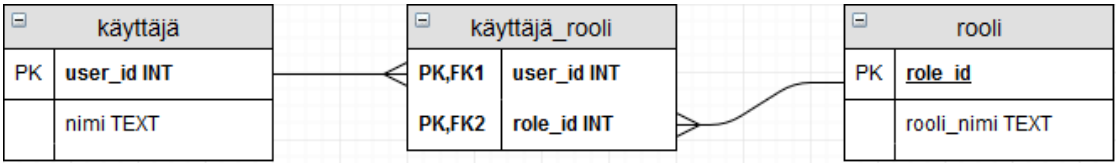
Kuvio 6. ER-kaavio 1:m -suhteella

ER-kaaviossa kahden taulun suhdetta kuvataan yhdistämällä taulut viivalla (Kuvio 6). Viiva voidaan vetää sarakkeiden välille, mutta laajoja tietokantoja mallinnettaessa viiva on selkeämpi piirtää taulusta tauluun.

2.7.1.1 Taulujen relaatioiden kardinaliteetti

Relaation kardinaliteetti kertoo, millaisesta suhteesta on kyse. Suhde voi olla One-To-Many, One-To-Zero-Or-One, Many-To-Many tai jokin näiden sekoitus. Viivan pää kertoo, millaisesta suhteesta on kyse. Kuviossa 6 näkyvä suhde on One-To-Many -suhde (1:m). One-To-Many -suhde piirretään viivalla, jonka Many -päässä on harava, One-päässä pelkkä viiva. Tämä tarkoittaa, että yhdellä komponentilla voi olla monta sarjanumeroa. Sama voidaan lukea myös toisinpäin: sarjanumeron pitää kuulua yhdelle ja vain yhdelle komponentille. Many-To-Many -relaatiota (m:n) voidaan käyttää, kun A voi viitata moneen B:hen, mutta myös B voi viitata moneen A:han. (Curry 2018.)

Tällainen suhde voisi olla taulut käyttäjistä ja käyttäjien rooleista. Yhdellä käyttäjällä voi olla monta roolia (admin, viewer, owner), mutta monella käyttäjällä voi olla sama rooli. Tällainen suhde toteutetaan yleensä luomalla kolmas taulu (ns. Join-table), jossa on sarakkeet kummankin taulun PK:lle. Taulut yhdistetään Join-tauluun One-To-Many -relaatiolla (Kuvio 7).



Kuvio 7. Many-To-Many -relaation toteutus Join-taululla

Käyttäjät yhdistetään rooleihin lisäämällä tietueita Join-tauluun (käyttäjä_rooli). Esimerkissä Join-taulun Primary Key:na ei käytetä yhtä saraketta. Tällainen ei toimisikaan, koska silloin tauluun ei voi tehdä toista tietuetta samalle käyttäjälle siten että Primary Key pysyisi uniikkina. Sen sijaan Join-taulun PK muodostetaan molemmista sarakkeista. Tällöin jokainen rivi on uniikki, sillä käyttäjälle tarvitsee merkata tietty rooli vain kerran.

käyttäjä

user_id	nimi
1	Esko Esimies
2	Seppo Selaaja

käyttäjä_rooli

user_id	role_id
1	1
1	2
2	2

rooli

role_id	rooli_nimi
1	admin
2	viewer

Kuvio 8. Roolien jako m:m -suhteella

Tässä esimerkissä (Kuvio 8) Eskolle on annettu admin ja viewer -oikeudet. Käyttäjä_rooli -taulussa on tietueet 1 – 1 sekä 1 – 2. Näillä Esko on yhdistetty rooleihin admin ja viewer. Sepolla on vain viewer -oikeudet, siispä 2 – 2.

2.8 CRUD

Akronyymillä CRUD tarkoitetaan tiedon tallentamisen perus funktioita. Järjestelmän (oli se sitten tietokanta tai REST API) tulisi pystyä suorittamaan seuraavat operaatiot:

- Create, uuden tietueen luonti
- Read, tiedon luku
- Update, tiedon päivitys
- Delete, tiedon poistaminen

(What is CRUD? n.d.)

2.8.1 Create (Insert)

MySQL-kielellä (ja myös muilla SQL-kielillä) uuden tiedon lisääminen tapahtuu Insert-käskyllä (Kuvio 9). Rivillä 2 määritetään taulu ja sarakkeet. Id-saraketta ei tarvitse tällä kertaa huomioida, sillä auto_increment huolehtii siitä. Values -avainsanan jälkeen kirjoitetaan tietueet sulkeiden sisään, toisistaan pilkuilla erotettuina. Tietueeseen on kirjoitettava arvoja yhtä monta, ja samassa järjestyksessä kuin sarakkeita on määritelty rivillä 2. Arvojen tietotyyppien on vastattava sarakkeiden tietotyyppiä. Tekstit kirjoitetaan ' -merkkien sisään. (INSERT Syntax n.d.)

```
1  INSERT INTO                                      # Lisätään Insert-käskyllä.
2      komponentti(nimitys, piirustusnumero)        # Taulu johon lisätään sekä
3                                                   # valitut sarakkeet.
4  VALUES
5      ('Iso planeetankantaja', 'GDRM101010'),      # Lisätään tietueet isolle ja pienelle
6      ('Pieni planeetankantaja', 'GDRM101328');    # kantajalle. Tietueet erotetaan pilkulla
```

Kuvio 9. Insert-käsky

2.8.2 Read (Select)

Tietokantaa luetaan Select-käskyllä. Käskylle on kerrottava mitä sarakkeita palautetaan ja mistä taulusta. Käsky tukee myös muita toimintoja, mutta sarakkeet ja taulu ovat ainoat pakolliset. Asteriski (*) valitsee kaikki taulun sarakkeet (Kuvio 10).

```
1 ●  SELECT                # Valitse
2        *                 # kaikki sarakkeet
3    FROM sarjanumero;     # sarjanumero-taulusta.
```

id	numero	pvm	komp_id
1	171354	2018-06-01	1
2	171355	2018-06-02	1
3	171026	2018-06-03	2

Kuvio 10. Select-käsky ja tulos

Jos asteriski korvataan luettelolla sarakkeita (Kuvio 11), tulokseen tulevat vain nämä sarakkeet. Suurien tietokantojen ja kyselyiden kanssa tämä vähentää tuloksen muodostamiseen tarvittavaa aikaa sekä verkon kuormaa. (SELECT Syntax n.d.)

```
1 ●  SELECT                # Valitse
2        numero,           # numero
3        pvm               # sekä päivämäärä
4    FROM sarjanumero;     # sarjanumero-taulusta.
```

numero	pvm
171354	2018-06-01
171355	2018-06-02
171026	2018-06-03

Kuvio 11. Select -käsky sarakkeittain

2.8.2.1 Where

Where-ehdolla voidaan määritellä mitkä taulun tietueet valitaan tulokseen. Sillä pystytään valitsemaan esimerkiksi vain täysi-ikäiset henkilöt, tai kadunnimet, jotka alkavat t-kirjaimella . Where-funktio ottaa parametrina predikaattifunktion, eli se on korkeamman asteen funktio (korkeamman asteen funktio on funktio, joka ottaa yhtenä parametrinaan tai antaa paluuarvonaan toisen funktion). (Perez 2015.)

Predikaattifunktio on funktio, jonka paluutyyppinä on boolean (Lee 2017). Where-funktion predikaattifunktion parametrina on oltava tietue. Palautettu boolean arvo määrittää otetaanko kyseinen tietue tulokseen.

```
1 ●  SELECT
2        *                          # Kaikki sarakkeet.
3    FROM sarjanumero
4    WHERE pvm < '2018-06-03';      # Tietueet, joissa pvm
5                                   # ennen kolmatta päivää.
```

id	numero	pvm	komp_id
1	171354	2018-06-01	1
2	171355	2018-06-02	1

Kuvio 12. Where-ehto

2.8.2.2 Join

Join -käskyllä (eri asia kuin Join-taulu) yhdistetään kaksi taulua relaatioistaan toisiinsa siten, että tulokseen voidaan etsiä tietoa molemmista tauluista. Yhdistyvillä sarakkeilla ei tarvitse olla Foreign Key-rajoitetta, mutta niillä on oltava sama tietotyyppi. (JOIN Syntax n.d.)

Ilman Join-käskyä, tiedon kysely kahdesta taulusta vaatii ylimääräisiä askelia. Ensin pitäisi tehdä kummallekin taululle omat kyselyt, ja yhdistää tulos halutuksi dataa käyttävässä sovelluksessa. Tämä on kuitenkin helpompaa (sekä tietokannalle nopeampaa) kun taulut yhdistetään Join-käskyllä.

Kun samassa kyselyssä käsitellään kahta tai useampaa taulua, sarakkeisiin viitataan syntaksilla:

taulu.sarake

jotta järjestelmä tietää kummasta taulusta sarakkeen tulisi löytyä. Myös aliaksia voidaan käyttää. Aliaksessa taululle annetaan helpompi ”lempinimi”, jota voidaan käyttää taulun kokonimen sijaan. Aliasten käyttö tekee pitkistä kyselyistä helpompia lukea. Taululle annetaan alias näin:

taulu t

ja sitä käytetään näin:

t.sarake

Otetaan esimerkkinä normalisoidut komponentti- ja sarjanumero-taulut (Kuvio 6). Relaatio on komponentin id-sarakkeesta sarjanumeron komp_id sarakkeeseen. Tulokseen halutaan sarjanumerot sekä kappaleen vastaava piirustusnumero ja nimitys.

```
1 •  SELECT
2        k.nimitys,              # Sarake-listassa käytetään aliaksia
3        k.piirustusnumero,
4        s.numero
5    FROM komponentti k          # Valitaan komponentti-taulu ja annetaan alias
6    JOIN sarjanumero s          # Yhdistetään komponentti-taulu sarjanumero-tauluun
7    ON k.id=s.komp_id;          # Yhdistävät sarakkeet. Aliasten käyttö myös täällä
```

Kuvio 13. Kysely kahdesta taulusta

Kysely (Kuvio 13) alkaa normaalisti valitsemalla halutut sarakkeet. Sarakkeet on merkattu aliaksilla. On hyvä huomata, että aliaksia voidaan käyttää ennen kuin niitä on määritelty.

Join-käsky yhdistää komponentti- ja sarjanumero-taulut. On-predikaatti määrittää millä taulujen sarakkeilla on relaatiot. Kyselyn tulokseen (Kuvio 14) tulee komponentin tiedot useampaan kohtaan, vaikka itse arvot on tallennettu vain kerran. Tämän takia normalisointi ja Join-käskyn käyttö nopeuttavat toimintaa sekä vähentävät redundanssia.

nimitys	piirustusnumero	numero
Iso planeetankantaja	GDRM101010	171354
Iso planeetankantaja	GDRM101010	171355
Pieni planeetankantaja	GDRM101328	171026

Kuvio 14. Kahden taulun kyselyn tulos

Join-käskyn voi myös ketjuttaa peräkkäin. Näin kysely voidaan suorittaa kolmelle tai useammalle taululle.

2.8.3 Update

Kolmas tiedon tallentamisen perusfunktioista on päivitys (Update). SQL-kielissä, Update -käskyllä valitaan päivitettävä taulu, asetetaan sarakkeille tietty arvo ja Where-predikaatilla valitaan päivitettävät tietueet. Update ilman where-predikaattia päivittää kaikkien tietueiden arvot. Jos haluaa päivittää vain yhden tietyn tietueen arvoja, where-predikaattiin on turvallisinta valita Primary Key (uniikkiutensa takia). (UPDATE Syntax n.d.)

Päivitetään komponentti-taulusta GDRM101010-komponentin nimitys Isoksi Planeettapyöräksi (Kuvio 15).

```
1 •  UPDATE komponentti                  # Päivitettävä taulu.
2    SET nimitys='Iso Planeettapyörä'    # Sarakkeen uusi arvo.
3    WHERE piirustusnumero='GDRM101010'; # Päivitettävät tietueet
```

Kuvio 15. Tiedon päivitys

Päivitettäviä sarakkeita voi olla enemmän kuin yksi. Tällöin uudet arvot kirjoitetaan pilkulla erotettuna:

sarake=uusi_arvo,

toinen_sarake=toinen_arvo

2.8.4 Delete

Tietue voidaan poistaa tietokannasta Delete-komennolla. Sille määritellään taulu, josta poistetaan, sekä halutessa where-predikaatti. Tämä on suositeltavaa, sillä muuten käsky tyhjentää koko taulun. Turvallisin tapa poistaa tietue on Primary Key:n perusteella poistaminen. Poistetaan kuvitteellisesta henkilö-taulusta Kalle-nimiset henkilöt, jotka ovat yli 30 vuotiaita (Kuvio 16).

```
1 •  DELETE FROM henkilö
2    WHERE etunimi='Kalle'
3            AND ikä>30;          # Predikaatti yhdistetty and-operaattorilla
```

Kuvio 16. Tietueen poistaminen

2.9 SQL-injektio

Paha-aikeinen käyttäjä voi vahingoittaa tietokantaa SQL-injektioksi kutsuttua hyökkäystapaa käyttäen. Yleinen käyttötapa tietokannalle jossain järjestelmässä on ottaa jokin käyttöliittymään syötetty arvo ja käyttää sitä parametrina kyselyn muodostamiseen. Käyttäjä voi esimerkiksi kirjoittaa tekstikenttään etunimen, ja saada tuloksena henkilöt, joilla on kyseinen nimi. Ohjelmakoodissa kysely voitaisiin muodostaa näin:

```
1. string query = "SELECT * FROM henkilö WHERE etunimi='" + firstName + "';";
```

Syötetty arvo lisätään suoraan kyselyyn. Käyttäjä voi syöttää tekstikenttään SQL-skriptin, jonka tietokanta suorittaa. Syöttämällä tekstikenttään skriptin:

'; DROP TABLE henkilö; --

kysely lähtee tietokantaan muodossa:

SELECT * FROM henkilö WHERE etunimi=''; DROP TABLE henkilö; -- ';

Tietokanta suorittaa molemmat käskyt. Ensimmäinen kysely valitsee henkilöt, joilta puuttuu etunimi (eli ei mitään vaarallista). Koska hyökkääjä lisäsi puolipilkun tekstin eteen, lopettaa se edellisen käskyn ja hyökkääjä voi antaa oman haitallisen käskynsä. Tässä tapauksessa hyökkääjä poistaisi koko henkilö-taulun. Haitallisen käskyn perään on lisätty viiva-viiva-välilyönti, jotta ohjelmakoodin loppuun lisäämät merkit kommentoituvat pois, ja tietokanta suostuu suorittamaan haitallisen käskyn.

2.9.1 Suojautuminen

SQL-injektiolta suojautuminen tapahtuu käsittelemällä käyttäjän syöttämät parametrit ohjelmakoodissa ennen kuin kysely lähetään tietokannalle. Ohjelman ulkopuolisia (eli käyttäjän syöttämiä) arvoja ei tulisi käyttää kyselyn muodostamiseen suoraan, vaan käyttää esimerkiksi tietokannan ohjelmarajapinnan toimintoja kyselyn parametroimiseen. Kyselyä ei muodosteta liittämällä ulkoiset arvot suoraan kyselyyn, vaan niiden sijaan käytetään paikkamerkkejä (engl. placeholder). Itse arvot lisätään jollain mulla tavalla rajapintaan ennen kyselyn suoritusta. Tällöin arvoa ei edes yritetä suorittaa, ja SQL-injektio ei ole mahdollista.

Parametrointi käytännössä riippuu käytetystä tietokannasta sekä ohjelmakoodin kielestä. Kaikilla on kuitenkin sama periaate taustalla. Kyselyn parametrointi C#-kielellä ja ADO.NET -komponenttikirjastolla:

```
1.  // Kyselyyn paikkamerkki parametrin kohdalle
2.  string query = "SELECT * FROM henkilö WHERE etunimi=?name";
3.  using(SqlCommand cmd = new SqlCommand(query, connection))
4.  {
5.      // Ulkoisen arvon parametrointi SqlCommand-olioon
6.      cmd.Parameters.AddWithValue("?name",  firstName);
7.
8.      // Kysely suoritetaan täällä
9.  }
```

SQL-kyselyn muodostuksessa käytetään paikkamerkkiä. Rivillä 3 luotu SqlCommand-olio kuvastaa suoritettavaa SQL-kyselyä tai -komentoa. Connection-muuttuja on SqlCommand -olio, joka määrittelee käytettävän yhteyden. Rivillä 6 parametrin arvo yhdistetään kyselyn paikkamerkkiin. Mahdollinen pahantahtoinen SQL-skripti käsitellään tällöin arvona, ja SQL-injektio epäonnistuu. (Bobby Tables: A guide to preventing SQL injection n.d.)

3 Full-Stack

Ohjelmisto voidaan jakaa front-end - ja back-end -osa-alueisiin. Front-end tarkoittaa käyttäjän käyttöliittymää sekä siihen liittyviä operaatioita. Back-end sijaitsee ”pellin alla”, ja se huolehtii ohjelmiston business-logiikasta. Yhdessä näistä muodostuu full-stack. (Anser 2018.)

Tässä kappaleessa käydään läpi tekniikoita ja aiheita, joita on hyödynnetty opinnäytetyön full-stack osuudessa.

3.1 C#

Järjestelmän logiikka kirjoitettiin C# kielellä (lausutaan C sharp). Se on Microsoftin kehittämä korkean tason ohjelmointikieli, ja syntaksiltaan se muistuttaa eniten muita C-kieliä ja Javaa. C# kielen tukemat ohjelmointiparadigmat:

- olio-ohjelmointi:
  - muuttujat ja metodit yhdistetään käsiteltäviksi kokonaisuuksiksi, joilla on tietty tyyppi
- geneerinen ohjelmointi:
  - voidaan kirjoittaa funktioita, joiden parametrien tai paluuarvojen tyypit tiedetään vasta ajon aikana
- imperatiivinen ohjelmointi:
  - tulokseen päästään suorittamalla järjestyksessä käskyjä, jotka muuttavat ohjelman tilaa
- funktionaalinen ohjelmointi:
  - funktion arvo saadaan sieventämällä lauseketta yksinkertaisemmaksi
- reflektiivinen kieli:
  - ohjelma pystyy tarkastelemaan ja muuttamaan omaa rakennettaan ja käyttäytymistään,
- rinnakkainen:
  - tuki monisäikeiseen suorittamiseen (multi-threading).

(Rouse n.d.; Bertrand 2018; Imperative programming 2017; Elliot 2017; Krauss 2016; Norvell 2009)

C# sai alkunsa Microsoftilla vuonna 1999, kun se kehitettiin korvaamaan .NET-luokkakirjastojen kirjoittamiseen käytetty SMC-kieli (Simple Managed C). Se tunnettiin aluksi nimellä Cool (C-like Object Oriented Language), mutta heinäkuussa 2000 .NET-kehyksen julkaisun yhteydessä sen nimeksi vaihtui C#. Nimi tulee musiikista, jossa ylennettyä nuottia merkitään risuaidalla; C# onkin ylemmän tason kieli kuin C tai C++. C#-kooditiedoston pääte on .cs. (The history of C# 2017.)

3.1.1 Kääntäjä

Toisin kuin MySQL, joka on tulkattava kieli, C# on käännettävä kieli. Ohjelman lähdekoodi käännetään ensin alustaneutraalille CIL-kielelle (Common Intermediate Language). Alustakohtainen CLR (Common Language Runtime) kääntää CIL-koodin konekieleksi, joka pystytään suorittamaan kyseisessä alustassa.

Käännettävän C#-koodin on oltava syntaksiltaan eheä, jotta kääntäjä suostuu kääntämään sen. Toisin kuin tulkattavissa kielissä, osa ohjelmointivirheistä löytyy jo tässä vaiheessa. (.NET Framework Platform Architecture 2015.)

3.1.2 Syntaksi

Syntaksiltaan C# muistuttaa muita C-kieliä ja Javaa. Käskyt lopetetaan puolipilkulla ja ryhmitellään aaltosulkeilla. Muuttujat ja metodit sijaitsevat luokissa, ja luokat nimiavaruudessa. Esimerkki staattisesta metodista, joka tulostaa perinteisen ”Hello, world!” -tekstin:

```
1.  using System;
2.
3.  namespace Foo
4.  {
5.      class Bar
6.      {
7.          static string message = "Hello, world!";
8.
9.          public static void Hello()
10.         {
11.             Console.WriteLine(message);
12.         }
13.     }
14. }
```

Rivillä 3 avataan nimiavaruus ’Foo’. Nimiavaruudella voidaan erotella samannimiset luokat toisistaan. Samassa nimiavaruudessa ei voi olla kahta samannimistä luokkaa. Nimiavaruudella voidaan helposti ryhmitellä vain tiettyyn toimintoon kuuluvat luokat johonkin toiseen toimintoon liittyvistä luokista.

Rivillä 5 alkaa ’Bar’-luokan määrittely. Luokan tulee sijaita joko nimiavaruudessa tai toisen luokan sisällä (nested class). Nimiavaruudessa voi olla useampia luokkia.

Muuttujat voidaan määrittää joko luokkaan tai suoraan metodiin. Teksti message olisi voitu määritellä myös ’Hello’-metodin sisään. Metodin sisällä määritelty muuttuja unohdetaan suorituksen poistuttua metodista.

Metodin ’Hello’ määrittely alkaa rivillä 9. Public-avainsana määrittää metodin näkyvyyden (scope). Static viittaa muistialueeseen. Void kertoo, että metodi ei anna paluuarvoa. Jos metodi antaisi palautusarvon, sen tyyppi kirjoitettaisiin void-sanan tilalle. Metodin nimen perään sulkeiden sisään kirjoitettaisiin metodin parametrien tyypit ja nimet. Vaikka ’Hello’-metodi ei otakkaan parametrejä, tyhjät sulkeet kirjoitetaan silti. Tällä erotetaan metodi attribuutista.

Rivillä 11 kutsutaan ’Console’-luokan ’WriteLine’-metodia, joka tulostaa parametrina annetun tekstin komentoriville. ’Console’-luokka löytyy ’System’-nimiavaruudesta, joka on osa .NET-ohjelmistokehystä. Nimiavaruus on otettu käyttöön rivillä 1. Ilman tätä riviä, kääntäjä etsisi ’Foo’-luokasta ’WriteLine’-nimistä metodia. Sellaista ei löydy, joten kääntäjä ei suostuisi kääntämään koodia suoritettavaksi binääritiedostoksi. Ilman ensimmäistä riviä, WriteLine-metodia kutsuttaisiin näin:

```
1. System.Console.WriteLine(message);
```

(Getting started: Hello, world! n.d.)

3.1.3 Olio-ohjelmointi

Object Oriented Programming (OOP) eli olio-ohjelmointi on ohjelmointiparadigma, jossa operaatiot ja attribuutit (kentät) yhdistetään loogisiksi kokonaisuuksiksi. Paradigman keskiössä on olio-konsepti. Olion ominaisuuksiin kuuluu, että sen operaatiot (metodit) muuttavat olion attribuuttien arvoja, ja näin ollen olion tilaa. Oliot ovat luokkien ilmentymiä, jotka määrittävät samalla olion tyypin. Toisin kuin proseduraalinen ohjelma, joka koostuu suoritettavista ohjeista, olio-ohjelma koostuu olioista, jotka toimivat keskenään.

Olio-ohjelmointi helpottaa ja nopeuttaa laajojen ja monimutkaisien ohjelmistojen kehitystä ja ylläpitoa. Se myös vähentää ohjelmointivirheitä ja redundanssia. Koska ohjelmakoodi yksittäisessä oliossa voi olla lyhyt, on sen ymmärtäminen ja korjaaminen helpompaa. (Rouse n.d.)

3.1.3.1 Olio

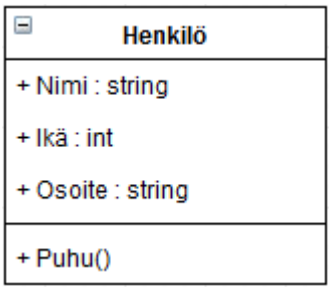
Olio (object) on ohjelman perusyksikkö. Sen tehtävänä on muistaa loogisesti toisiinsa liittyvien muuttujien arvot, ja sillä on operaatioita, jotka muuttavat näitä arvoja. Operaatio (metodi) on suoritettava ohjelman osa, joka käynnistyy kutsuttaessa. Metodi ottaa syötteen, manipuloi dataa ja generoi paluuarvon. C# tukee metodeja, jotka eivät ota syötettä tai anna paluuarvoa. (Petkov 2018.)

3.1.3.2 Luokka

Luokkaan määritellään olion ominaisuudet. Se on kuin olion pohjapiirustus. Olio on aina jonkin luokan ilmentymä. (Petkov 2018.)

3.1.3.3 Unified Modelin Language

Unified Modelin Language (UML) on Object Management Groupin (OMG) ylläpitämä visuaalinen merkintäkieli, jolla voidaan mallintaa ja suunnitella järjestelmän eri toimintoja. Se on kehitetty vuosina 1994-1995, ja se otettiin osaksi ISO-standardia 2005. Vaikka UML:ään kuuluu monia eri kaavioita (jotka mallentavat eri ominaisuuksia), tässä työssä keskitytään luokkadiagrammiin.



Kuvio 17. Luokkadiagrammi

Luokkaa kuvataan laatikolla, jonka yläosassa lukee luokan nimi (Kuvio 17). Seuraavaksi luetellaan luokan attribuutit. Attribuutin syntaksi:

[näkyvyys] [attribuutin_nimi] : [tyyppi]

Näkyvyys voi olla:

- public (+),
- private (-),
- protected (#),
- internal (~).

Julkinen (public) attribuutti näkyy muille luokille. Yksityinen (private) näkyy vain samassa luokassa. Suojeltu (protected) attribuutti on kuin yksityinen, mutta se näkyy myös aliluokassa. Sisäinen (internal) ei näy ohjelmakirjaston (assembly) ulkopuolelle.

Attribuuttien jälkeen määritellään luokan jäsenfunktiot (metodit). Jäsenfunktion määrittelyn syntaksi:

[näkyvyys] [funktion_nimi]([parametrit]) : [paluuarvon tyyppi]

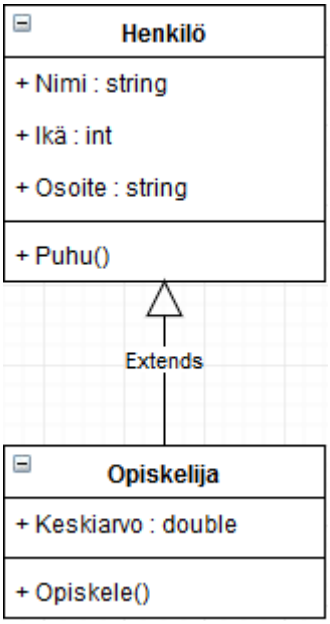
jossa parametri:

[parametrin_nimi] : [parametrin tyyppi]

Parametrit erotetaan pilkulla. (UML Class Diagram Tutorial n.d.)

3.1.3.4 Perintä

Luokat voivat periä ominaisuuksia muilta luokilta. Luokalla ’Henkilö’ voi olla aliluokka ’Opiskelija’. Tällöin ’Henkilö’-luokan attribuutit (joita voisi olla esim. ’Nimi’, ’Ikä’ ja ’Osoite’) periytyvät myös ’Opiskelija’-luokalle. ’Opiskelija’-luokalla voi olla myös muita attribuutteja, kuten ’Keskiarvo’, mutta nämä eivät vaikuta ’Henkilö’-luokkaan. ’Opiskelija’-luokan tyypiksi tulee ’Opiskelija’, mutta sillä on myös tyyppi ’Henkilö’.



Kuvio 18. Perintä olio-ohjelmoinnissa

UML-kielessä perintää merkitään avoimella nuolella (Kuvio 18). Nuoli vedetään perivästä luokasta perittävään luokkaan. (UML Class Diagram Tutorial n.d.)

’Henkilö’-luokan ja siitä perivän ’Opiskelija’-luokan määrittely C#-kielellä:

```
1.  // määritellään Henkilö-luokka
2.  class Henkilö
3.  {
4.      public string Nimi {
5.          get;
6.          set;
7.      }
8.
9.      public int Ikä {
10.         get;
11.         set;
12.     }
13.
14.     public string Osoite {
```

```
15.         get;
16.         set;
17.     }
18.
19.     public void Puhu() {
20.         Console.WriteLine("Hei, nimeni on {0}", Nimi);
21.     }
22.
23. }
24.
25. // Opiskelija perii Henkilö-luokan
26. class Opiskelija : Henkilö
27. {
28.     public double Keskiarvo {
29.         get;
30.         set;
31.     }
32.
33.     public void Opiskele()
34.     {
35.         // Opiskeluun liittyvä koodi
36.     }
37. }
```

Rivillä 26 aloitetaan 'Opiskelija'-luokan määrittely. Kaksoispisteen jälkeen kirjoitetaan luokka, jonka jäsenet halutaan periä.

Moniperinnästä puhutaan silloin, kun luokka perii ominaisuuksia monelta eri luokalta. Toisin kuin C++, C# ei tue moniperintää, vaan hierarkkista perintää, jossa luokalla voi olla vain yksi superluokka.

3.1.3.5 Polymorphismi

Oliota, joka perii jonkin yläluokan jäsenet, voidaan käyttää samoissa yhteyksissä kuin sen yläluokan oliotakin. Tämä on mahdollista, koska aliluokalla on myös yläluokan tyyppi. Esimerkissä on funktio 'Käsittele', joka ottaa parametrina käsiteltävän henkilön. Funktiolla voidaan siis käsitellä myös 'Opiskelija'-luokan olioita, sillä ne ovat myös 'Henkilö'-tyyppisiä. (Petkov 2018.)

```
1.  void Käsittele(Henkilö henkilö)
2.  {
3.      // henkilön käsittely...
4.  }
5.
6.  // uusi Henkilö-luokan ilmentymä
7.  Henkilö henkilö = new Henkilö();
8.  Käsittele(henkilö);
9.
10. // uusi opiskelija
11. Opiskelija opiskelija = new Opiskelija();
12. Käsittele(opiskelija); // opiskelija-olio parametrinä
13.
14. // ei sallittu parametri, int ei ole Henkilö-tyyppinen
15. int value = 5;
16. Käsittele(value); // generoi kääntäjän errorin
```

3.1.4 Muistialueet

Muuttujat voidaan määritellä kolmelle eri muistialueelle: stack, heap, static. Muistialue vaikuttaa siihen, miten ja missä muuttujaa voi käyttää. (Skeet n.d.)

3.1.4.1 Stack

Muuttuja tallentuu stack-alueelle eli pinkkaan silloin, kun se määritellään metodin sisällä. Tämä tarkoittaa sitä, että muuttuja siivotaan automaattisesti pois muistista, kun suoritus poistuu metodista. Muuttujaa voi käyttää ainoastaan metodin sisällä, mutta sen arvo voidaan toki antaa paluuarvona takaisin. (Skeet n.d.)

Muuttujan määrittely ja käyttö stack-alueella:

```
1.  void Foo()
2.  {
3.      string message = "Hello, world!";
4.      System.Console.WriteLine(message);
5.  }
```

Koska muuttujan näkyvyys on vain metodin sisällä, samannimisen muuttujan määrittely jossain toisessa metodissa on sallittua:

```
1.  void DoSomething()
2.  {
3.      int value = 5;
4.      // loput metodista...
5.  }
6.
7.  void DoSomethingElse()
8.  {
9.      int value = 7;
10.     // loput metodista...
11. }
```

3.1.4.2 Heap

Muuttuja tallennetaan heap-alueelle, kun se määritellään luokkaan, muttei metodin sisälle. Tällöin muuttujan arvo säilyy, vaikka suoritus poistuisikin luokan metodista. Tällaista muuttujaa kutsutaan instanssimuuttujaksi, sillä se pysyy muistissa yhtä kauan kuin luokan ilmentymä. Saman luokan instanssimuuttujaan viitataan this-avainsanalla. Tämä on kuitenkin pakollista vain, jos metodissa halutaan käyttää samannimista pinkkaan tallennettavaa muuttujaa. (Skeet n.d.)

Instanssimetodin määrittely ja käyttö Foo-luokassa:

```
1.  class Foo
2.  {
3.      string bar = "Hello, world!";
4.
5.      void Hello()
6.      {
```

```
7.          Console.WriteLine(this.bar);
8.      }
9.  }
```

Julkisen instanssimuuttujan käyttö toisessa luokassa:

```
1.  class Foo // luokka
2.  {
3.      // julkinen instanssimuuttuja
4.      public int Bar = 5;
5.  }
6.
7.  // ... jossain muualla:
8.  Foo foo = new Foo();
9.  Console.WriteLine(foo.Bar);
```

Instanssimuuttujan käyttöön tarvitaan aina olio, jolle se kuuluu.

3.1.4.3 Static

Kolmas muistialue on static. Tämän muistialueen muuttujat kuuluvat aina tyypille, eivätkä yksittäiselle ilmentymälle. Staattinen muuttuja määritellään kuten instanssimuuttuja, mutta static-avainsanalla. (Skeet n.d.)

Muuttujaa käytetään tyypin nimellä, ei oliolla:

```
1.  class Foo
2.  {
3.      public static int Bar = 5;
4.  }
5.
6.  // muualla...
7.  Console.WriteLine(Foo.Bar);
```

3.2 Windows Forms

Windows Forms (WinForms) on luokkakirjasto graafisen käyttöliittymän (GUI, Graphical User Interface) luomiseen. Se on osa .NET-ohjelmistokehystä. WinFormsin jälkeen on julkaistu muita GUI-kirjastoja (kuten WPF ja UWP), mutta WinForms on vielä laajassa käytössä yksinkertaisuutensa vuoksi.

WinForms on tapahtumapohjainen (event-driven) teknologia, jossa ohjelman ajoon vaikuttavat tapahtumat (events) sekä näiden delegaatit (event handlers), jotka suorittavat halutut käskyt. Esimerkiksi Tallenna-painikkeen Click-tapahtuma liipaistaan painiketta painettaessa, ja tämän tapahtuman delegaattimetodi (esimerkiksi BtnSave_Click) suorittaa tarvittavat toimenpiteet tiedon tallentamiseen.

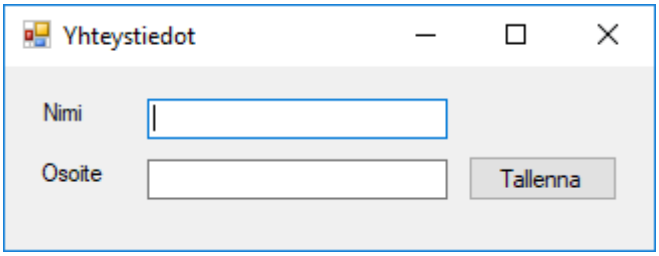
Käyttöliittymä WinFormsilla voidaan rakentaa joko raahaamalla kontrollit ruudulle, tai proseduraalisella koodilla, esimerkiksi ikkunan alustajassa. (Windows Forms 2017.)

3.2.1 Yhteystiedot-esimerkki

Esimerkissä luodaan käyttöliittymä henkilön nimen ja osoitteen tietokantaan tallentamista varten. WinFormsissa Form-luokka tarkoittaa ikkunaa. Tästä perityn Form1-luokan alustajassa luodaan ikkunaan aseteltavat kontrollit ja niiden ominaisuudet. BtnSave-painikkeen Click-tapahtuman event-handler on luotu anonyymilla metodilla. Anonyymillä metodilla ei ole nimeä, ja se voidaan määritellä toisen metodin sisällä lambda-kalkyylista tutulla merkitsemistavalla, jossa parametrien tyyppejä ei tarvitse merkitä. Kääntäjä pystyy päättelemään käytetyt tyypit kontekstista.

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();

        // ikkunan koko
        this.Width = 340;
        this.Height = 130;

        // ikkunan otsikko
        this.Text = "Yhteystiedot";

        // nimi-label
        Label lblName = new Label()
        {
            Width = 35,
            Location = new Point(16, 16),
            Text = "Nimi",
        };

        // nimi-tekstikenttä
        TextBox txtName = new TextBox()
        {
            Width = 150,
            Location = new Point(71, 16),
        };

        // osoite-label
        Label lblAddress = new Label()
        {
            Width = 45,
            Location = new Point(16, 46),
            Text = "Osoite",
        };

        // osoite-tekstikenttä
        TextBox txtAddress = new TextBox()
        {
```

```
            Width = 150,
            Location = new Point(71, 46),
        };

        // tallenna-painike
        Button btnSave = new Button()
        {
            Width = 75,
            Location = new Point(231, 44),
            Text = "Tallenna",
        };

        // tallenna-painikkeen event handler
        btnSave.Click += (sender, eventArgs) =>
        {
                // tekstikenttien arvot
                string name = txtName.Text;
            string address = txtAddress.Text;

                // Save-metodin kutsu
                DBClient.Save(name, address);

                // Ilmoitus käyttäjälle
                MessageBox.Show($"Nimi: {name}\nOsoite: {address}",
                "Tallennettu");
        };

        // lisätään kontrollit ikkunan kontrolleihin
        this.Controls.AddRange(new Control[]
        {
                lblName,
                txtName,
                lblAddress,
                txtAddress,
                btnSave,
        });
    }
}
```



Kuvio 19. Yhteystiedot-käyttöliittymä

DBClient on staattinen luokka, joka hoitaa tietokantaan tallentamisen. Sen yksityinen ConnectionString-kenttä määrittää käytettävän yhteyden tietokantaan.

```
public static class DBClient
{
    // connection string yhteyden muodostamiseen
    private static readonly string ConnectionString =
        "server=localhost;database=HumanResourcesDB;" +
        "uid=root;password=salainen_salasana";
```

```
    public static void Save(string name, string address)
    {
        // yhteys tietokantaan
        using (MySqlConnection connection =
            new MySqlConnection(ConnectionString))
        {
            // suoritettava SQL-skripti
            string query = "INSERT INTO contacts (name, address) " +
                           "VALUES (?name, ?address);";

            // SQL-komento
            using (MySqlCommand cmd = new MySqlCommand(query, connection))
            {
                // komennon parametrointi
                cmd.Parameters.AddWithValue("?name", name);
                cmd.Parameters.AddWithValue("?address", address);

                // yhteyden avaus
                connection.Open();

                // komennon suoritus
                cmd.ExecuteNonQuery();
            }
        }
    }
}
```

Save-metodi ottaa parametreina henkilön nimen ja osoitteen. Ensimmäisenä luodaan MySqlConnection-objekti, joka edustaa yhteyttä tietokantaan. ConnectionString-kenttää käytetään sen alustajassa. Seuraavaksi kirjoitetaan käytettävä SQL-komentoteksti. Komentoon ei suoraan liitetä parametrien arvoja, vaan käytetään placeholder-nimiä. Placeholder-nimi alkaa ?-merkillä. Komentotekstillä ja MySqlConnection-objektilla voidaan luodaa MySqlCommand-objekti, joka edustaa suoritettavaa komentoa. Lisättävän nimen ja osoitteen arvot lisätään parametreina vasta tässä vaiheessa, kutsumalla komento-objektin Parameters.AddWithValue-metodia. Näin menettelemällä vältytään SQL-injektiolta. (Bobby Tables: A guide to preventing SQL injection n.d.)

Nyt voidaan avata yhteys ja suorittaa komento. Yhteyttä ei tarvitse erikseen sulkea, koska MySqlConnection-luokka toteuttaa IDisposable-rajapintaa, ja sitä kutsutaan using-lohkossa.

3.2.1.1 IDisposable-rajapinta ja using-lohko

Rajapinta kertoo luokan käyttäytymisestä. Rajapintaan määritellään mitä metodeja, funktioita ja attribuutteja toteuttavalla luokalla tulee vähintään olla, mutta näiden implementointi riippuu vain luokasta itsestään. IDisposable-rajapinta määrittää, että toteuttavalla luokalla tulee olla parametroimaton Dispose-metodi. Dispose-metodin tulisi vapauttaa käytetyt resurssit, esimerkiksi edellä käytetty MySqlConnection-luokan Dispose-metodissa suljetaan käytetty yhteys.

Using-lohkon sulkujen sisällä luodaan objekti, jonka luokka toteuttaa IDisposable-rajapintaa. Objektia käyttävät kutsut kirjoitetaan aaltosulkeiden sisään (objektin näkyvyys on rajoitettu aaltosulkeiden sisään). Objektin Dispose-metodia kutsutaan sulkevan aaltosulkeen kohdalla automaattisesti. Using-lohkoa on hyvä käyttää IDisposable-rajapintaa toteuttavien luokkien kanssa, sillä se vähentää ohjelmointivirheiden riskiä. (Using objects that implement IDisposable 2017.)

4 Toteutus

Työssä käytettiin Visual Studio 2015 -ohjelmistoa luokkakirjaston sekä sovellusten kirjoittamiseen. Tietokanta luotiin MySQL Workbench 6.3-ohjelmistolla.

Koska järjestelmä perustuu kerättyjen tietojen käsittelyyn, ensimmäisenä luotiin GearClassLibrary-luokkakirjasto. Tällä tavoin luotiin business-logiikka, joka antaa raamit kaikelle muulle toiminnalle.

Seuraavaksi luotiin tietokanta. Tietokannan suunnittelussa arvioitiin luotua luokkakirjastoa, ja luotiin tarvittavat taulut tiedon tallennukseen. Tietokannan relaatiot luotiin huolehtimaan tallennetun datan eheydestä. Yleisenä hyvänä käytäntönä pidetään, että datan eheydestä huolehditaan siellä missä data on tallennettu, eikä tietokantaa käyttävässä sovelluksessa (Martinez n.d.).

Viimeisenä luotiin sovellukset käyttöliittymineen yhdistämään logiikan ja tietokannan käyttäjiin. Ikkunoiden asettelun ja toiminnallisuuden suunnittelussa tukeuduttiin henkilöstön väliseen dialogiin. Erilaisia ratkaisuja koitettiin, ja valmiiseen järjestelmään valittiin toimivimmat.

4.1 Lähdekooditiedostot

Tässä osassa on lueteltu järjestelmän lähdekoodin tiedostot. Jokainen tiedosto vastaa samannimistä luokkaa.

GearClassLibrary:

- Assembly.cs
- Bearing.cs
- CheckList.cs
- Component.cs

- DB.cs
- DetailedAssembly.cs
- ExcelMethods.cs
- Gear.cs
- GuideLink.cs
- INPC.cs
- LeanDB.cs
- Measurement.cs
- MeasurementCalculator.cs
- MeasurementCalculatorExpression.cs
- MeasurementLogging.cs
- NotTraceablePart.cs
- Part.cs
- Plan.cs
- PlanetUnit.cs
- Report.cs
- ReportScheme.cs
- ResourceProvider.cs
- SerialNumberLogging.cs
- ShowOnUIAttribute.cs
- SubAssemblySelection.cs
- ToolLogging.cs
- Work.cs
- WorkPhase.cs
- WrongComponentTypeException.cs

Assembly Manager:

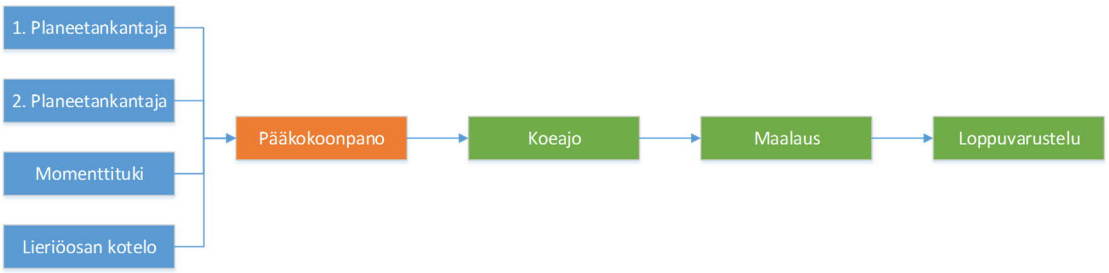
- AssemblyWindow.cs
- CalculatorExpressionWizard.cs
- CheckItemInputBox.cs
- Form1.cs
- GearTemplateWindow.cs
- GuideInput.cs
- LeanCredentialForm.cs
- LeanDatabase.cs
- MeasurementDetails.cs
- MySqlDatabase.cs
- NewAssemblyWindow.cs
- PartDetails.cs
- Program.cs
- ReorderMeasurementsWindow.cs
- ReorderPartsWindow.cs
- ReportForm.cs
- SelectAssemblyWindow.cs
- SelectNotTraceableParts.cs
- SelectTemplateWindow.cs
- ToolDetails.cs
- WorkHeaderInputBox.cs
- WorkPhaseWizard.cs

Assembly Reporter:

- Controls
  - BearingPanel.cs
  - CalculationPanel.cs
  - ComponentPanel.cs
  - GenericPanel.cs
  - MeasurementGroupBox.cs
  - PlanetUnitPanel.cs
  - SerialNumberLoggingGroupBox.cs
  - SubAssemblySelectionGroupBox.cs
  - ToolLoggingGroupBox.cs
  - ToolPanel.cs
  - WorkListPanel.cs
  - WorkPanel.cs
  - WorkPhasePanel.cs
- Models
  - IVerifiable.cs
  - MeasurementUpdatedEventArgs.cs
  - WorkBoxItem.cs
  - WorkPhaseNotValidEventArgs.cs
- CheckListDgv.cs
- ImageForm.cs
- LeanDatabase.cs
- Main.cs
- MySqlDatabase.cs
- Program.cs
- SelectAssemblyWindow.cs
- ShowAllMeasurements.cs
- ShowAllParts.cs
- ShowLink.cs
- WWRForm.cs

5 Tuuliturbiinivaihteen kokoonpanoprosessi

Tuuliturbiinivaihteen kokoonpanoprosessi koostuu alikokoonpanoista, sekä nämä yhdistävästä pääkokoonpanosta. Pääkokoonpanon jälkeen vaihde vielä koeajetaan, maalataan sekä loppuvarustellaan.

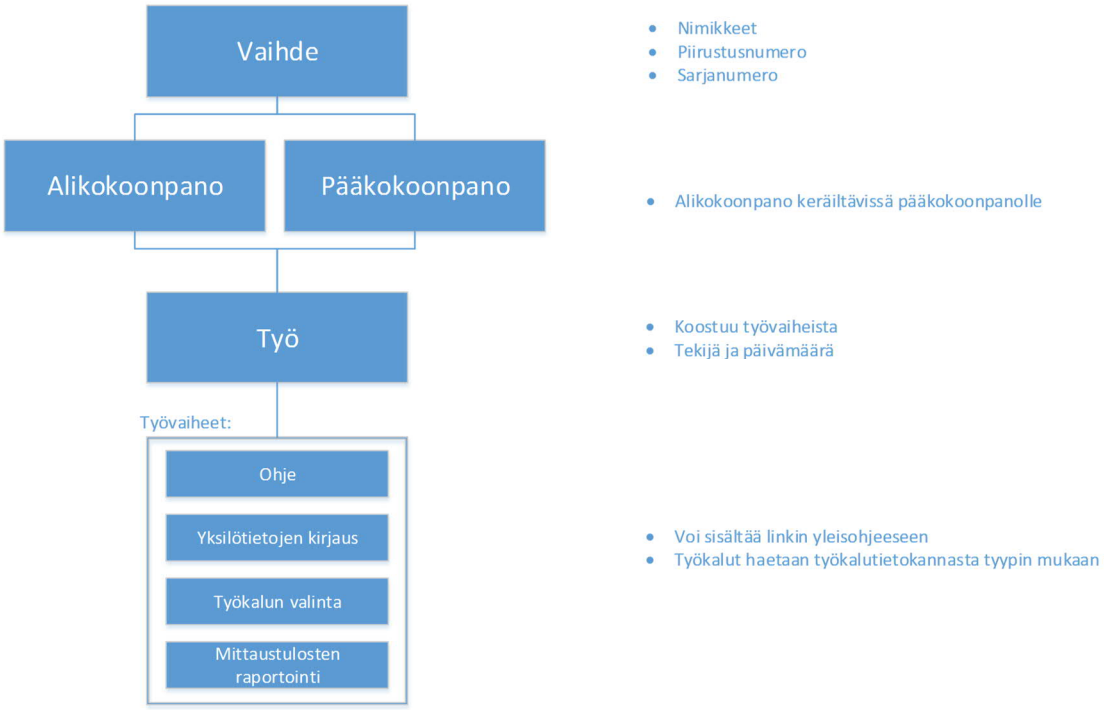


Kuvio 20. Kokoonpanoprosessi

5.1 Kokoonpanoprosessin ohjaus sekä raportointi

Kokoonpanoprosessia ohjataan vaiheittain näytettävillä työohjeilla, jotka samalla pyytävät asentajaa kirjaamaan halutut tiedot. Tällaisia tietoja on mm. asennetun komponentin sarjanumero, käytetty työkalu tai mittaustulos.

Kuten mainittu, jokainen kokoonpanoprosessi koostuu pääkokoonpanosta, sekä tämän alikokoonpanoista. Nämä puolestaan koostuvat töistä, jotka koostuvat työvaiheista. Työvaihe on yksityiskohtaisin osa kokoonpanoprosessista. Esimies kohdistaa jäljitystiedon kirjaamisen yksittäiselle työvaiheelle.



Kuvio 21. Vaihteen kokoonpanoprosessin ohjeistuksen rakenne

Vaihteella on tyyppi, nimike, piirustusnumero, versio sekä välitys. Vaihteelle kohdistetaan kokoonpanot. Vain yksi kokoonpano voi olla pääkokoonpano. Kun pääkokoonpano valmistuu, kyseinen vaihde merkitään valmiiksi. Kokoonpanolla on nimi suomeksi ja englanniksi, nimike, sekä piirustusnumero. Alikokoonpanolla tulee olla sarjanumero, jotta pääkokoonpanossa voidaan valita oikea alikokoonpano asennettavaksi. Alikokoonpanon sarjanumeroksi tulee jokin alikokoonpanon tietyn komponentin sarjanumero. Pääkokoonpanon sarjanumero haetaan tuotannonohjausjärjestelmästä tehdyn työnumeron perusteella.

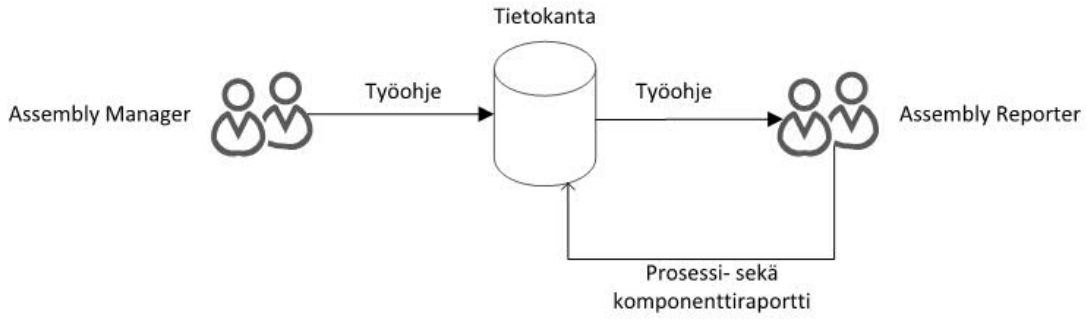
Työ kohdennetaan valmistettavaksi tietyllä työpisteellä. Tuotannon puolella työntekijä työpisteeen valitsemisen jälkeen näkee vain sillä työpisteellä valmistettavat kokoonpanot.

Työ koostuu työvaiheista. Työvaiheella on aina työohje suomeksi ja englanniksi, ja se voi sisältää linkin oppaaseen, ohjekuvan, listan asennettavista komponenteista joiden jäljistystietoja ei kerätä (pultit, mutterit…) tai esimerkiksi keskiarvoistuksen aiemmin mitatuista mittaustuloksista. Työvaiheella on lisäksi tyyppi, jolla voidaan lisätä toiminnallisuutta työohjeeseen. Työvaihteen tyypit ovat:

- Työohje
  - Pelkkä työohjeen teksti sekä mahdolliset kuvat ja linkit oppaisiin
- Yksilötietojen kirjaus
  - Kirjataan asennettujen komponenttien jäljitystiedot
- Mittaustulosten kirjaus
  - Kirjataan mittaustulokset, esimerkiksi kriittinen välysmitta.
  - Mittauksilla voi olla toleranssit, joiden ylitys näkyy asentajalle punaisella.
- Käytetyn työkalun kirjaus
  - Valitaan käytetty työkalu. Työkalut tyyppeineen haetaan toiminnanohjausjärjestelmästä.
- Alikokoonpanon keräily
  - Valitaan asennettu alikokoonpano
- Tarkistulista
  - Yksi tai useampi kohta, jotka asentaja tarkastaa.

5.2 Arkkitehtuuri

Järjestelmä koostuu tietokannasta, GearClassLibrary-luokkakirjastosta sekä näitä käyttävistä Assembly Manager- ja Assembly Reporter-sovelluksista. Assembly Manager alustaa GearClassLibraryn luokkien ilmentymät halutuilla arvoilla, ja tallentaa ne tietokantaan. Assembly Reporter tulkitsee tietokannasta haetut objektit, ja näyttää ne asentajalle. Asentajan kirjaamat arvot tallennetaan takaisin tietokantaan.



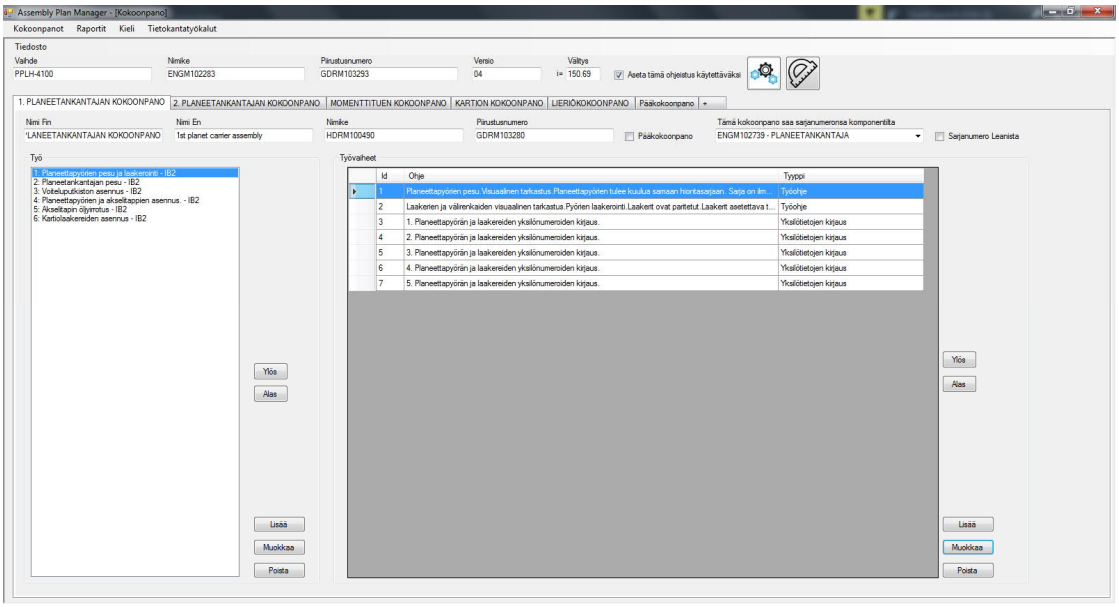
Kuvio 22. Järjestelmän työkierto

6 Sovellukset

6.1 Assembly Manager

Esimiehen työkalu on Assembly Manager-sovellus. Sillä luodaan kokoonpano-ohjeet vaihteelle, sekä määritetään käytettävät työkalut, kirjattavat jäljistytiedot sekä mittaukset ja niiden toleranssit. Ohjeet tallennetaan tietokantaan, josta Assembly Reporter hakee aina uusimman. Assembly Managerilla myös hallinnoidaan itse tietokantaa. Sillä voidaan poistaa vanhoja (tai muuten vain tarpeettomia) ohjeita, sekä ylläpitää listaa työpisteistä, laakerivalmistajista sekä valmistusmaista, sekä luodaan raporttipohjat komponenttiraporteille.

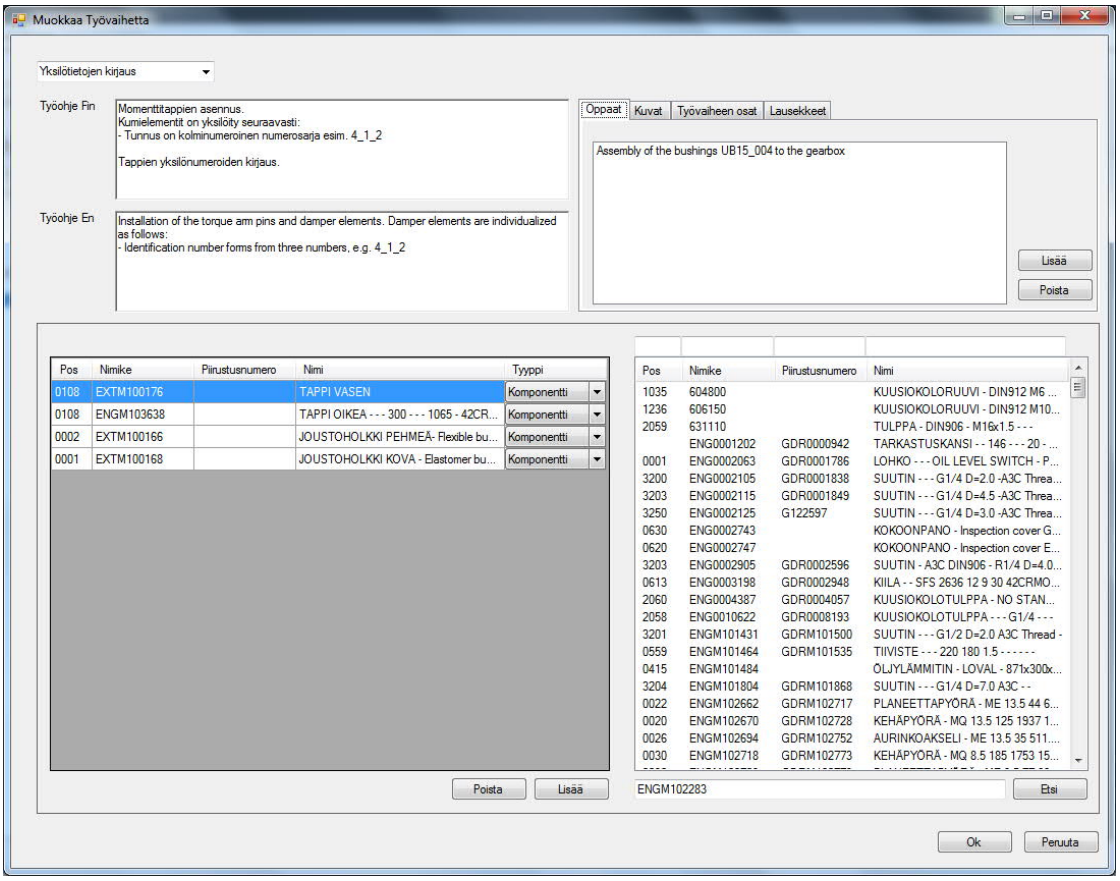
6.1.1 Yleisnäkymä



Kuvio 23. Assembly Manager yleisnäkymä

Assembly Managerin yleisnäkymässä (Kuvio 23) näkyy ylhäällä vaihteen yleiset tiedot. Välilehdet edustavat kokoonpanoja. Välilehden vasemmassa laidassa on listattuna työt sekä työpisteen koodi, joille se on kohdennettu. Keskellä näkyy valitun työn työvaiheet sekä sen tyyppi.

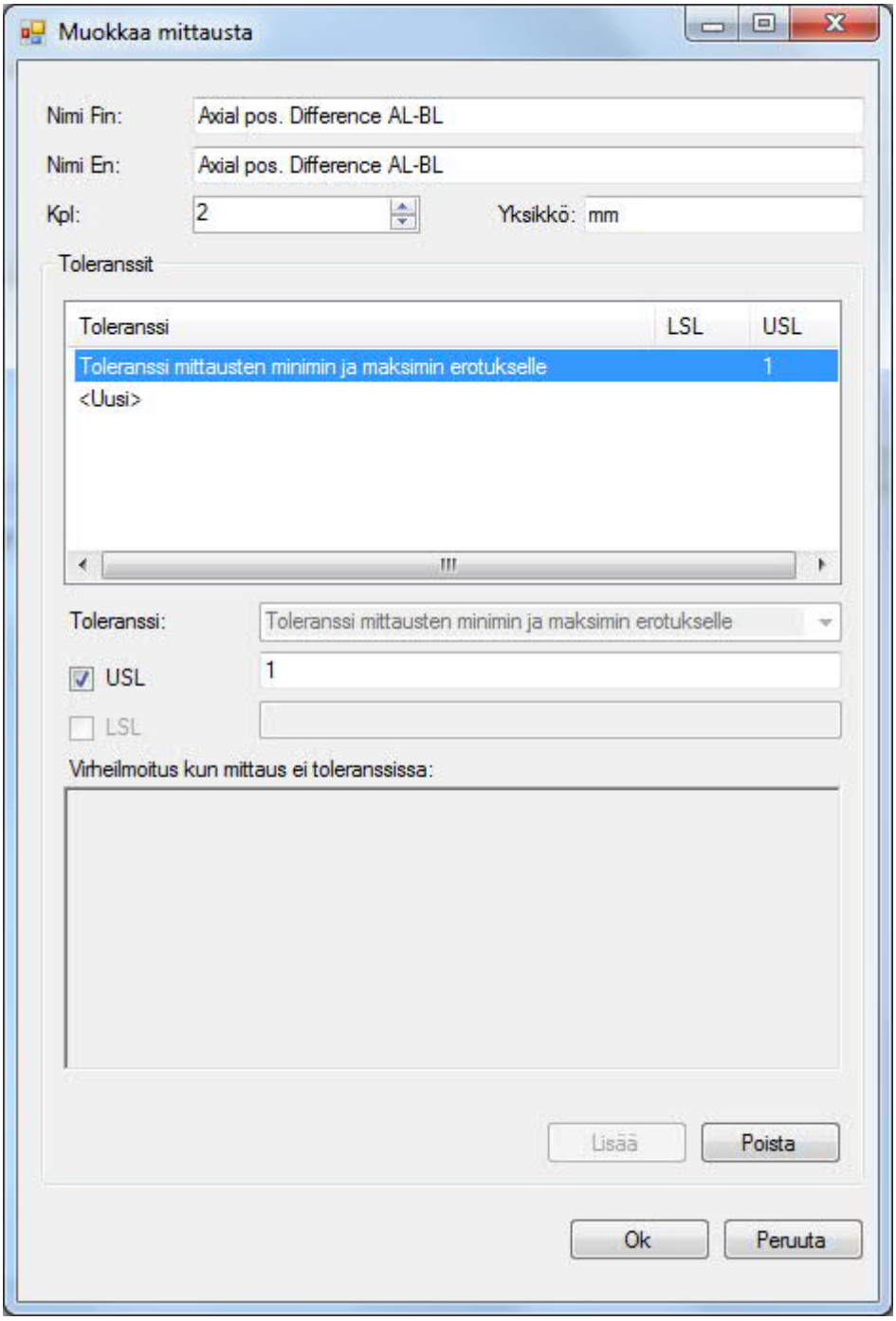
6.1.2 Yksilötietojen kirjauksen määritys



Kuvio 24. Näkymä työvaiheen muokkauksesta

Työvaihe määritellään Kuvio 24 mukaisella näkymällä. Vasemmasta yläkulmasta valitaan haluttu tyyppi. Tekstikenttiin kirjoitetaan työohjeet suomeksi ja englanniksi. Oikealle yläkulmaan voidaan määrittää näytettävä oheistieto, kuten kuva tai opas. Alavasemmalla on lista komponenteista, joiden jäljitystiedot kerätään. Komponentin tyyppi voi olla joko komponentti, planeettapyörä tai laakeri. Valittu tyyppi määrittää, mitä jäljitystietoja kerätään. Oikealla alhaalla näkyy lista kaikista kyseisen vaihdetyypin komponenteista. Lista haetaan toiminnanohjausjärjestelmästä, ja sitä voidaan käyttää ohjeen luomisen nopeuttamiseksi.

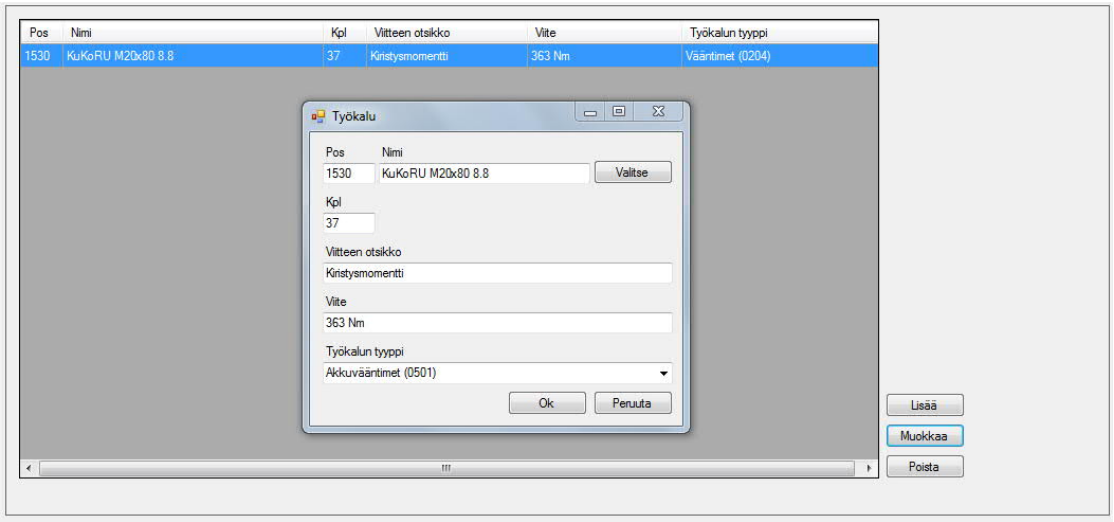
6.1.3 Mittauksen toleranssin määrittely



Kuvio 25. Näkymä mittauksen toleranssin määrittelyyn

Kuvio 25 mukaisella näkymällä määritellään mittauksen nimi, kappalemäärä tuloksista, yksikkö sekä sen toleranssit. Mittaukselle voi myös määrittää virheilmoituksen, jos mittaus ei ole toleranssissa.

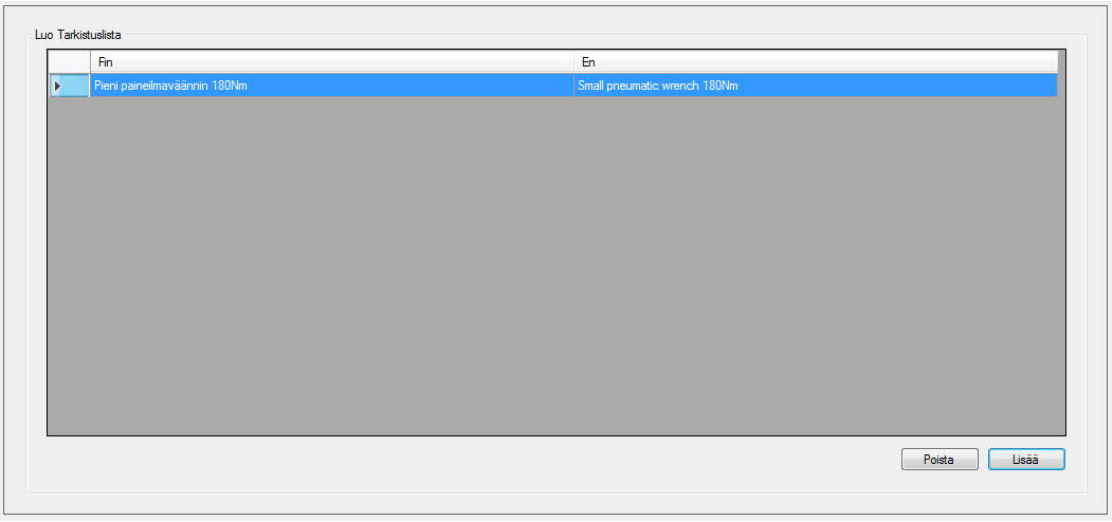
6.1.4 Käytetyn työkalun kirjaus



Kuvio 26. Käytetyn työkalun kirjauksen määrittely

Kuvio 26 mukaisella näkymällä määritellään käytetty työkalu. Sille valitaan osa ja sen positio (esimerkiksi mutteri tai pultti), kappalemäärä, otsikko, viite sekä tyyppi. Työkalut tyyppeineen ja sarjanumeroineen on kirjattu toiminnanohjausjärjestelmään. Tuotannossa asentaja pystyy valitsemaan vain oikean tyyppisen työkalun.

6.1.5 Tarkistuslista



Kuvio 27. Tarkistulistan luominen

Tarkistuslistaan lisätään tarkistettavat kohdat suomeksi ja englanniksi.

6.1.6 Apunäkymät

Assembly Managerissa on myös kaksi apunäkymää, jotka auttavat kokoonpano-ohjeistuksen luomisessa. Toinen näyttää kaikki vaihteeseen asennettavat komponentit (Kuvio 28), ja toinen kaikki vaihteen mittaukset (Kuvio 29).

Kaikki osat - voit muuttaa osien järjestystä raahaamalla rivejä

Nimike	Piirustusnumero	Nimi	Tyyppi
ENGM102777	GDRM102840	5. AKSELITAPPI	Komponentti
ENGM102777	GDRM102840	3. AKSELITAPPI	Komponentti
ENGM102777	GDRM102840	2. AKSELITAPPI	Komponentti
ENGM102739	GDRM102797	PLANEETANKANTAJA	Komponentti
ENGM102662	GDRM102717	4. PLANEETTAPYÖRÄ	Planeettapyörä
ENGM102662	GDRM102717	3. PLANEETTAPYÖRÄ	Planeettapyörä
ITMM200833		4. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ENGM102662	GDRM102717	2. PLANEETTAPYÖRÄ	Planeettapyörä
ITMM200833		3. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ENGM102662	GDRM102717	1. Planeettapyörä	Planeettapyörä
ITMM200833		2. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ENGM102662	GDRM102717	5. PLANEETTAPYÖRÄ	Planeettapyörä
ITMM200833		1. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ITMM200833		5. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ENGM102836	GDRM102903	LIERIÖPORTAAN KOTELO	Komponentti
ITMM200834		SOVITETTU KARTIORULLALAAKERIPARI	Laakeri
ENGM102777	GDRM102840	4. AKSELITAPPI	Komponentti
ENGM102777	GDRM102840	1. AKSELITAPPI	Komponentti
ENGM102723	GDRM102779	1. PLANEETTAPYÖRÄ	Planeettapyörä
ITMM200836		1. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ENGM102723	GDRM102779	2. PLANEETTAPYÖRÄ	Planeettapyörä
ENGM102723	GDRM102779	3. PLANEETTAPYÖRÄ	Planeettapyörä
ITMM200836		2. SOVITETTU LIERIÖRULLALAAKERIPARI	Laakeri
ENGM102738	GDRM102796	VÄLIKARTIO	Komponentti
ENGM102718	GDRM102773	KEHÄPYÖRÄ	Komponentti
ENGM102778	GDRM102841	2. AKSELITAPPI	Komponentti
ENGM102694	GDRM102752	AURINKOAKSELI	Komponentti
ENGM102778	GDRM102841	3. AKSELITAPPI	Komponentti
ENGM102778	GDRM102841	1. AKSELITAPPI	Komponentti
ENGM102744	GDRM102804	PLANEETANKANTAJA	Komponentti

Kuvio 28. Näkymä kaikista osista

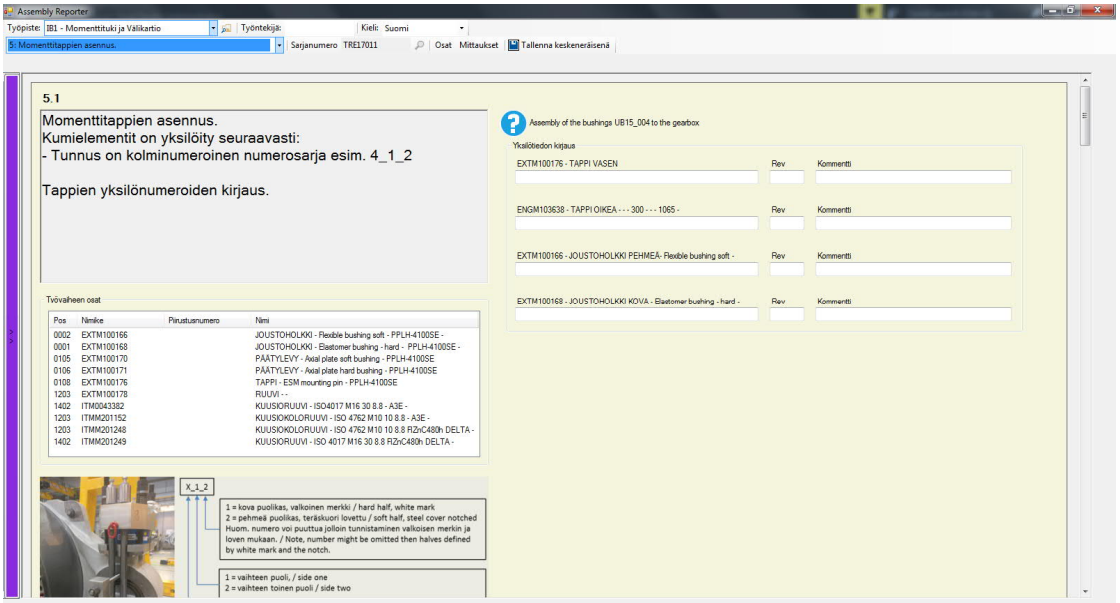
Kaikki mittaukset - voit muuttaa mittausten järjestystä raahaamalla rivejä

Nimi	Kpl	Toleranssi
Sovitesegmentin paksuus	1	True
T3 Nopean akselin heitto	1	True
Laakeri-kehäpyörä	1	False
Välikartio T9	1	False
Mittakello 1, kierros 1	1	False
Mittakello 2, kierros 1	1	False
Mittakello 3, kierros 1	1	False
Mittakello 1, kierros 2	1	False
Mittakello 2, kierros 2	1	False
Mittakello 3, kierros 2	1	False
Mittakello 1, kierros 3	1	False
Mittakello 2, kierros 3	1	False

Kuvio 29. Näkymä kaikista mittauksista

6.2 Assembly Reporter

Tuotannon puolella asentaja käyttää Assembly Reporter-sovellusta kokoonpano-ohjeiden lukemiseen ja jäljitystietojen kirjaamiseen. Työpisteen ja kokoonpanon valinnan jälkeen sovellus näyttää työohjeet työ kerrallaan. Työ valmistuu vasta kun kaikkiin sen työvaiheisiin on kirjattu halutut tiedot. Töitä voi kuitenkin selata eteenpäin ilman, että edellinen on valmistunut. Jokaiseen työvaiheeseen on mahdollista kirjata kommentti.



Kuvio 30. Assembly Reporter yleisnäkymä

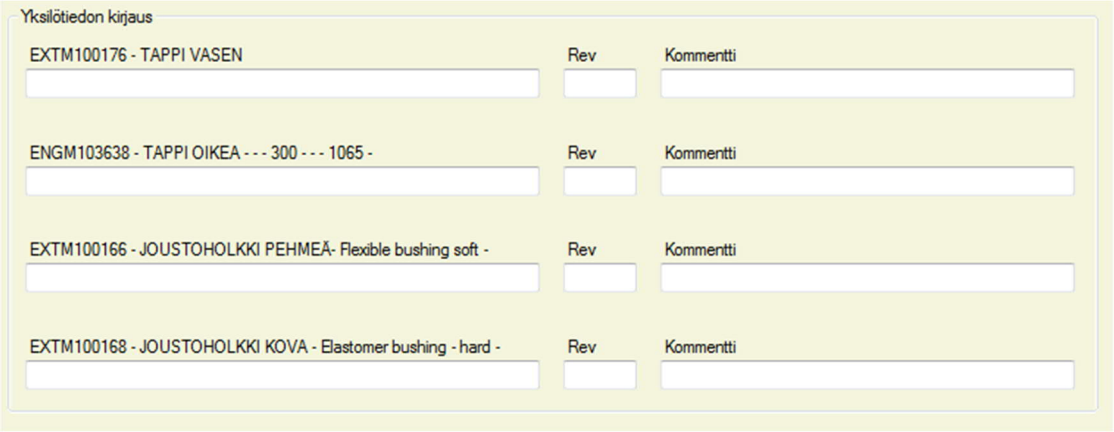
Työvaiheen osat on lueteltu työohjeen alapuolella (Kuvio 31). Nämä ovat siis osia, jotka asennetaan työvaiheessa, mutta joiden yksilötietoja ei kirjata. Tämä toimii asentajan muistilistana. Jos työvaiheeseen on liitetty kuva, se näytetään työvaiheen osien alapuolella (kuva aukeaa isommaksi klikkaamalla).

Työvaiheen osat

Pos	Nimike	Piirustusnumero	Nimi
0002	EXTM100166		JOUSTOHOLKKI - Flexible bushing soft - PPLH-4100SE -
0001	EXTM100168		JOUSTOHOLKKI - Elastomer bushing - hard - PPLH-4100SE -
0105	EXTM100170		PÄÄTYLEVY - Axial plate soft bushing - PPLH-4100SE
0106	EXTM100171		PÄÄTYLEVY - Axial plate hard bushing - PPLH-4100SE
0108	EXTM100176		TAPPI - ESM mounting pin - PPLH-4100SE
1203	EXTM100178		RUUVI - -
1402	ITM0043382		KUUSIORUUVI - ISO4017 M16 30 8.8 - A3E -
1203	ITMM201152		KUUSIOKOLORUUVI - ISO 4762 M10 10 8.8 - A3E -
1203	ITMM201248		KUUSIOKOLORUUVI - ISO 4762 M10 10 8.8 FlZnC480h DELTA -
1402	ITMM201249		KUUSIORUUVI - ISO 4017 M16 30 8.8 FlZnC480h DELTA -

Kuvio 31. Työvaiheen osat

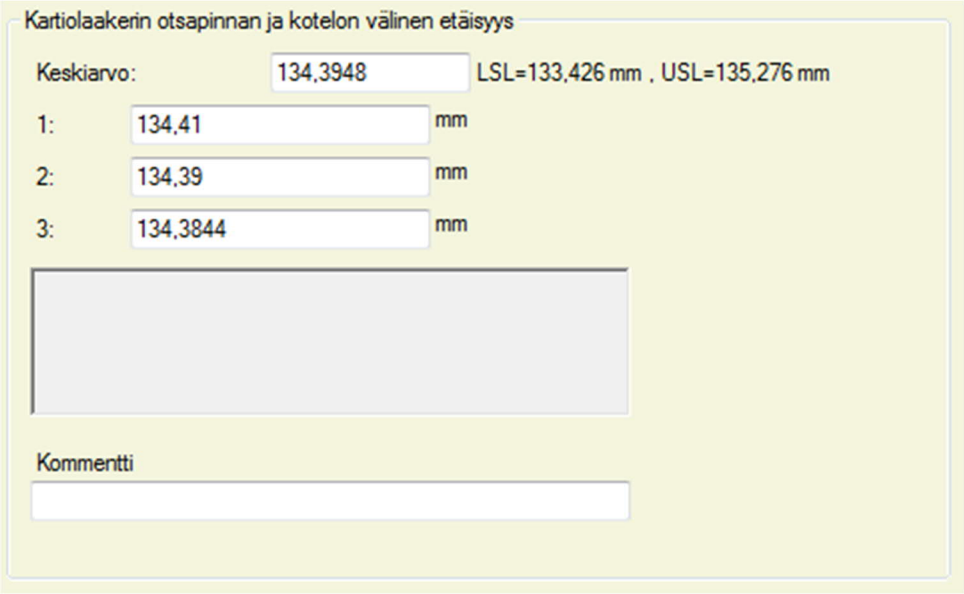
Yksilötietojen kirjaus tapahtuu työohjeiden oikealla puolelle sijoitettuihin tekstikenttiin (Kuvio 32, Kuvio 33). Työvaihetta ei saa valmistettua, ennen kuin joka kohtaan on kirjattu vähintään sarjanumero, sekä lisäksi planeettapyörille hiontaluokka ja laakereille valmistaja sekä valmistusmaa.



Kuvio 32. Yksilötietojen kirjaus - komponentit

Kuvio 33. Yksilötietojen kirjaus - planeettapyörä ja laakeri

Jos työvaiheen tyyppi on mittaustulosten kirjaus, kirjattavien komponenttien paikalla pyydetään mittaustulosta (Kuvio 34). Jos mittaustulos ei ole määritettyjen toleranssien sisällä, ilmoitetaan siitä käyttäjälle punaisella värillä. Tällöin työtä ei voida merkitä valmiiksi, ennen kuin poikkeama on kommentoitu, tai saatu uusi hyväksytty tulos.



Kuvio 34. Mittaustulosten kirjaus

Työkalun kirjaus näyttää asennettavan osan nimen, position, kappalemäärän sekä työkaluryhmän. Työkaluryhmästä riippuen voidaan näyttää myös lisätietoa, kuten kiristysmomentti (Kuvio 35). Alasvetovalikosta valitaan käytetty työkalu. Päivitä-painikkeella päivitetään työkalulista, jos työkalu ei ollut kyseisellä työpisteellä työtä aloitettaessa.

Kuvio 35. Työkalun kirjaus

7 Tulokset

Opinnäytetyön tavoitteena oli luoda kokoonpanoprosessin dokumentointijärjestelmä, jolle voidaan luoda työvaihekohtaisia ohjeita. Päätavoitteina oli luoda järjestelmä, joka olisi reaaliaikainen, helposti päivitettävissä, helppokäyttöinen ja joustava (voidaan soveltaa eri vaihdemalleille). Järjestelmästä muodostui hyvin toimiva kokonaisuus. Käyttäjiltä saatu palaute oli positiivista, etenkin kun järjestelmä yhdisti jäljitystietojen kirjauksen työohjeisiin ja oppaisiin (ennen opinnäytetyötä samat asiat sijaitsivat eri järjestelmissä).

7.1 Kehitettävää

Käyttöliittymä toteutettiin WinForms-teknologialla. Vaikka tämä on aivan toimiva ratkaisu, nykyaikaisempi tapa olisi ollut käyttää WPF-teknologiaa (Windows Presentation Foundation), ja MVVM-ohjelmistomallia (Model-View-ViewModel). Nämä ratkaisut olisivat helpottaneet järjestelmän päivittämistä ja jatkokehittämistä.

Työohjeet ja kirjatut jäljitystiedot tallennetaan sarjoittamalla sovelluksen käsittelemät objektit binäärilausekkeina tietokantaan. Tällöin tiedon haku ja analysointi hidastuu, ja vaatii aina GearClassLibrary-luokkakirjaston implementointia. Parempi tapa olisi ollut normalisoida tietokanta viimeistä komponenttia myöten, jolloin muiden järjestelmien käyttö datan analysointiin olisi helpompaa.

Lähteet

.NET Framework Platform Architecture. 20.7.2015. Microsoftin dokumentti. Viitattu 23.11.2018. https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework#net-framework-platform-architecture

Anser, M. Front-end vs Back-end. Which and Why. 2.4.2018. Blogikirjoitus. Viitattu 9.11.2018. https://medium.com/@ansertechgeek/front-end-vs-back-end-which-and-why-31c2a6b96b2c

Bertrand, C. Coding Concepts - Generics. 27.6.2018. Viitattu 16.11.2018. https://dev.to/designpuddle/coding-concepts---generics-34cf

Bobby Tables: A guide to preventing SQL injection. N.d. Viitattu 9.11.2018. http://bobby-tables.com/adodotnet

Curry, C. Cardinality and Modality. 14.8.2018. Opas binäärirelaatioihin kirjoittajan sivustolla. Viitattu 26.10.2018. https://www.calebcurry.com/cardinality-and-modality/

Elliot, E. Master the JavaScript Interview: What is Functional Programming? 4.1.2017. Blogikirjoitus Medium-sivustolla. Viitattu 16.11.2018. https://medium.com/javascript-scene/master-the-javascript-interview-what-is-functional-programming-7f218c68b3a0

Getting started: Hello, world! N.d. C# verkko-opas. Viitattu 23.11.2018. https://csharp.net-tutorials.com/getting-started/hello-world/

Imperative programming. 12.7.2017. Määritelmä Computer Hope -sivustolla. Viitattu 16.11.2018. https://www.computerhope.com/jargon/i/imp-programming.htm

INSERT Syntax. N.d. MySQL 8.0 Reference Manual. Viitattu 2.11.2018. https://dev.mysql.com/doc/refman/8.0/en/insert.html

Introduction to SQL. N.d. SQL verkko-opas. Viitattu 12.10.2018. https://www.w3schools.com/sql/sql_intro.asp

JOIN Syntax. N.d. MySQL 8.0 Reference Manual. Viitattu 9.11.2018. https://dev.mysql.com/doc/refman/8.0/en/join.html

Krauss, A. Programming Concepts: Type Introspection and Reflection. 12.2.2016. Artikkeli kirjoittajan sivustolla. Viitattu 23.11.2018. https://thesocietea.org/2016/02/programming-concepts-type-introspection-and-reflection/

Lee, X. Computer Language: Predicate Function, Terminology, and Naming Convention. 18.11.2017. Opas kirjoittaja sivustolla. Viitattu 2.11.2018. http://xahlee.info/comp/naming_of_predicate.html

Machajewski, S. N.d. What are Databases? - Examples & Types. Study.com verkkokurssi. Viitattu 12.10.2018. https://study.com/academy/lesson/what-are-databases-examples-types-quiz.html. Types of Databases.

Martinez, F. N.d. Bad practices in Database Design: Are you making these mistakes? Viitattu 2.12.2018. https://www.toptal.com/database/database-design-bad-practices. Bad Practice No. 4: Bad Referential Integrity (Constraints)

Moventas Gears Oy. 2018. About us. Viitattu 12.10.2018. https://www.moventas.com/about-moventas/

Mundy I. Declarative vs Imperative Programming. 20.2.2017. Blogikirjoitus codeburst.io -sivustolla. Viitattu 12.10.2018. https://codeburst.io/declarative-vs-imperative-programming-a8a7c93d9ad2

Normalization of Database. N.d. Tietokantajärjestelmäopas. Viitattu 26.10.2018. https://www.studytonight.com/dbms/database-normalization.php

Norvell, T. What is Concurrent Programming? 9.1.2009. Luentomoniste. Viitattu 23.11.2018. https://www.engr.mun.ca/~theo/Courses/cp/pub/cp0.pdf

Perez, N. Higher order functions, what are they? 10.12.2015. Blogikirjoitus Hackernoon-sivustolla. Viitattu 2.11.2018. https://hackernoon.com/higher-order-functions-what-are-they-be74111659e8

Petkov, A. How to explain object-oriented programming concepts to a 6-year-old. 27.6.2018. Blogikirjoitus Medium-sivustolla. Viitattu 30.11.2018. https://medium.freecodecamp.org/object-oriented-programming-concepts-21bb035f7260

PHP MySQL Database. N.d. PHP verkko-opas. Viitattu 12.10.2018. https://www.w3schools.com/php/php_mysql_intro.asp. PHP: MySQL Database.

Rouse, M. Definiton: Object-oriented programming. N.d. Viitattu 16.11.2018. https://searchmicroservices.techtarget.com/definition/object-oriented-programming-OOP

Rouse, M. & Biscobing J. Definition: Relational Database. N.d. Osa Evolution of Windows Azure SQL Database -opasta. Viitattu 26.10.2018. https://searchdatamanagement.techtarget.com/definition/relational-database

SELECT Syntax. N.d. MySQL 8.0 Reference Manual. Viitattu 2.11.2018. https://dev.mysql.com/doc/refman/8.0/en/select.html

Skeet, J. Memory in .NET - what goes where. N.d. Artikkeli kirjoittajan sivustolla. Viitattu 30.11.2018. http://jonskeet.uk/csharp/memory.html

SQL - Primary Key. N.d. SQL verkko-opas. Viitattu 19.10.2018. https://www.tutorialspoint.com/sql/sql-primary-key.htm

SQL Syntax. N.d. SQL verkko-opas. Viitattu 19.10.2018. https://www.w3schools.com/sql/sql_syntax.asp

The history of C#. 20.9.2017. Microsoftin documentti. Viitattu 23.11.2018. https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

UML Class Diagram Tutorial. N.d. Opas Lucidchart-yrityksen sivustolla. Viitattu 30.11.2018. https://www.lucidchart.com/pages/uml-class-diagram

UPDATE Syntax. N.d. MySQL 8.0 Reference Manual. Viitattu 9.11.2018. https://dev.mysql.com/doc/refman/8.0/en/update.html

Using objects that implement IDisposable. 7.4.2017. Dokumentti Microsoftin sivustolla. Viitattu 1.12.2018. https://docs.microsoft.com/en-us/dotnet/standard/garbage-collection/using-objects

What is CRUD? N.d. Artikkeli CodeAcademy-sivustolla. Viitattu 26.10.2018. https://www.codecademy.com/articles/what-is-crud

What is ERD? N.d. Entity Relation Diagram verkko-opas. Viitattu 19.10.2018. https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/

Windows Forms. 30.3.2017. Dokumentti Microsoftin sivustolla. Viitattu 1.12.2018. https://docs.microsoft.com/en-us/dotnet/framework/winforms/