Otto Koivisto

TOPOLOGY C# TESTAUS JA KEHITYS

Tietojenkäsittelyn koulutusohjelma

2016

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TOPOLOGY C# TESTAUS JA KEHITYS

Koivisto, Otto
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Toukokuu 2016
Ohjaaja: Grönholm, Jukka
Sivumäärä: 41

Asiasanat: C#, Visual Studio, ohjelmointi, SQL

Tämä opinnäytetyö toteutettiin Teollisuuden Voima Oyj:lle. Aiheena oli Topology-ohjelmiston tilannekartoitus ja edelleen kehittäminen. Topology ja sen käsittelemä putkistotietokanta ovat osa suurempaa PAMS nimistä järjestelmää. PAMS on lyhenne sanoista TVO Piping and Component Analysis and Monitoring System.

Topology ohjelmistoa kehitettiin Microsoftin tarjoamilla työkaluilla. Kehitysympäristönä toimi Visual Studio 2012, ohjelmointikielenä C# ja ohjelmistokehyksenä .NET.

Alkuun suoritetun tilannekartoituksen jälkeen kehitystä pyrittiin viemään mahdollisimman pitkälle aikataulun puitteissa aloittaen korkeimman prioriteetin tehtävistä.

Tuloksena opinnäytetyöstä saatiin selvempi kuva projektin tilasta ja kaksi suurta kokonaisuutta saatettiin siihen pisteeseen, että ne ovat valmiita laajempaan käyttäjätestaukseen. Vaikka opinnäytetyö onkin saatettu päätökseen, jatkuu Topology-ohjelmiston kehittäminen edelleen.

TOPOLOGY C# TESTING AND DEVELOPMENT

Koivisto, Otto
Satakunta University of Applied Sciences
Degree Programme in Business Information Technology
May 2016
Supervisor: Grönholm, Jukka
Number of pages: 41

Keywords: C#, Visual Studio, programming, SQL

This Bachelor's thesis was commissioned by Teollisuuden Voima Oyj. The purpose of the project was to map out the status of the software development project of the software called Topology and to develop the software further. Topology and the piping database it handles are both part of a system called PAMS, which stands for TVO Piping and Component Analysis and Monitoring System.

Development was done with tools provided by Microsoft. Visual Studio 2012 was used as an integrated development environment. The programming language used was C# and .NET provided the framework.

After mapping out the status of the project the focus was on taking the development as far as possible within the allotted time starting with the highest priority tasks.

As a result the status mapping yielded a clearer view of the status of the program and two major components of the program were developed to a point at which they are ready for larger scale user testing. Although this thesis has been concluded, the development of Topology is still ongoing.

TERMIT JA LYHENTEET

API
Lyhenne sanoista Application Programming Interface. Se on määritelmä, jonka perusteella eri ohjelmat voivat vaihtaa tietoa keskenään yhteisesti sovituilla menetelmillä.

C#
Projektissa käytetty Microsoftin kehittämä oliopohjainen ohjelmointikieli.

CAD
Lyhenne sanoista englanninkielen sanoista Computer Aided Design. Se tarkoittaa tietokoneen käyttöä apuvälineenä yleisesti insinöörien ja arkkitehtien suunnittelutyössä.

LINQ
Lyhenne sanoista Language-Integrated Query. .NET-komponentti, joka mahdollistaa datakyselyt.

.NET
Microsoftin kehittämä ohjelmistokomponenttikirjasto, jonka päälle voidaan kehittää monenlaisia ohjelmistoja.

OLE DB
Lyhenne sanoista Object Linking and Embedding Database. Kyseessä on Microsoftin kehittämä API (ohjelmointirajapinta), joka mahdollistaa mm. tietokantojen ohjelmallisen lukemisen ja kirjoittamisen.

VB
Visual Basic ohjelmointikieli

Visual Studio 2013
Microsoftin kehittämä kehitysympäristö useille ohjelmointikielille, joista yleisimmin käytetyt ovat C# ja VB.NET.

WinForms
Lyhenne sanoista Windows Forms. .NET komponenttikirjastoon sisältyvä luokkakirjasto. Sitä käytetään graafisten käyttöliittymien luomiseen Windows ympäristössä.

SISÄLLYS

1 JOHDANTO

Opinnäytetyö tehtiin Teollisuuden Voima Oyj:lle. Työn valvojana toimi lujuuslaskentapäällikkö Paul Smeekes. Hänen lisäkseen työtä ohjasi ohjelmistokonsultti Olli Kuuluvainen.

Työn aiheena oli Topology C# nimisen ohjelmistoprojektin tilannekartoitus ja jatkokehitys. Tavoitteena oli saattaa ohjelma ainakin osittaiselle beta-testausasteelle ja myös kehittää ohjelmaa tavalla, joka palvelisi tulevaa kehitystyötä. Tähän pyrittiin rakentamalla ohjelman dokumentointia ja koodin huolellisella kommentoinnilla. Putkistotietokantojen tiedon eheyden säilyttäminen on hyvin tärkeää, joten työssä oli kiinnitettävä erityistä huomiota ohjelmiston luotettavaan toimintaan testaamalla jokainen osa-alue perusteellisesti.

Topologyä käytetään putkistotietokannan lukemiseen ja muokkaamiseen. Topology on luonteeltaan hyvin visuaalinen ohjelma ja kykenee luomaan kolmiulotteisia malleja putkistojen rakenteista tietokannan datan perusteella. Topologystä on olemassa aiempi Visual Basic 6.0 ympäristöllä toteutettu versio. Microsoft lopetti Visual Basic 6.0 tuen vuonna 2008, jonka vuoksi Topologyn toteuttamiselle modernimmassa ympäristössä oli tarvetta. C# versiota oli kehitetty jo pitkään ennen tämän opinnäytetyön alkua ja kehittäjiä oli ollut lukuisia. Projektin edetessä ja kehittäjien vaihtuessa kokonaiskuva projektin statuksesta oli hämärtynyt ja tämän vuoksi tilanteeseen haluttiin selvyyttä tilannekartoituksen avulla.

2 TYÖKALUT JA MENETELMÄT

2.1 Visual Studio 2013

Topology kehitys tapahtui pääasiassa Visual Studiolla, joka on Windows-ympäristössä toimiva Microsoftin kehittämä ohjelmistokehitysympäristö. Sillä voidaan kehittää ohjelmistoja Windows, web ja myös mobiiliympäristöihin. Visual Stu-

dio sisältää kattavan työkaluvalikoiman ja sen tärkeimpiä komponentteja ovat koodieditori, IntelliSense (koodin automaattinen täydennys), koodin refaktorointityökalut ja debuggeri. Ohjelmasta löytyy myös graafisten käyttöliittymien suunnitteluun tarkoitettu koodikirjasto. (Microsoft 2016a)

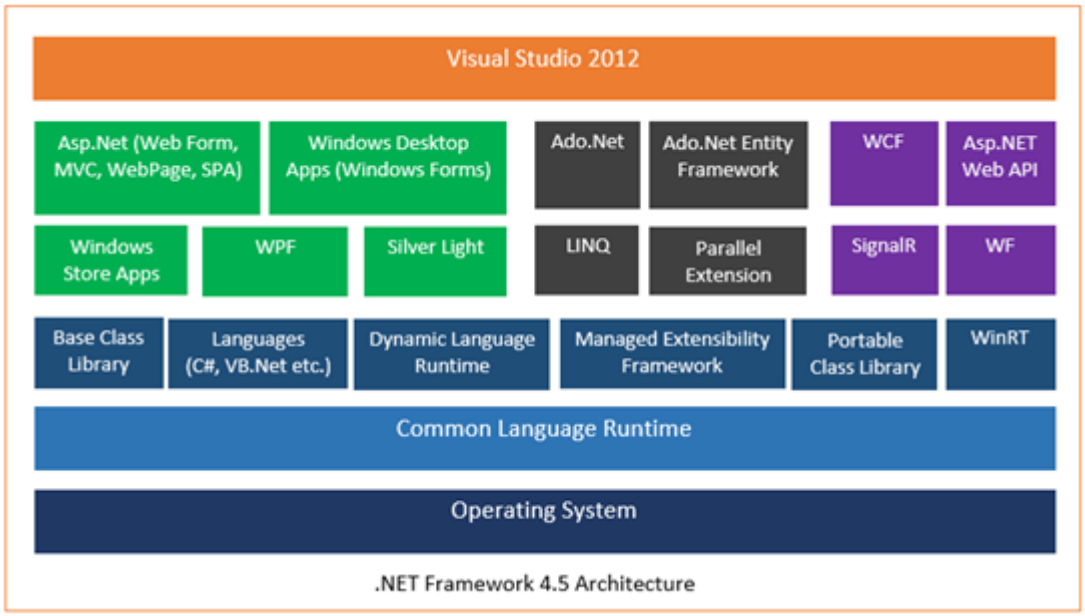
Visual Studiosta löytyy sisäänrakennettuna tuki C, C++, VB.NET C# ja F# ohjelmointikielille. Tuet muille kielille kuten M, Python, Ruby ja monet muut ovat myös mahdollisia erikseen asennettavien lisäpalveluiden myötä. Tärkeä piirre Visual Studiossa on sen ”plug in” periaatteella toimiva laajennettavuus. Ohjelman toiminnollisuutta pystytään laajentamaan helposti asentamalla joko Microsoftin tai jonkun muun tekijän kehittämillä lisäpaketeilla. Paketteja pystyy myös itse tarvittaessa luomaan ja käyttämään monissa eri projekteissa. Näin käyttäjä ei ole koskaan sidottu ainoastaan tiettyyn valikoimaan toiminnollisuutta. (Microsoft 2016a) Tätä toiminnollisuutta käytettiin Topology projektissa mm. lisäämällä Visual Basic Powerpacks luokkakirjasto, joka sisältää tarpeellisia piirtotyökaluja.

2.2 .NET Framework

.NET Framework on Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota Microsoft Visual Studio-ympäristössä kehitetyt ohjelmistot käyttävät. Se tukee noin 20 ohjelmointikieltä ja kaikki sen tukemat kielet ovat sen myötä vaihdettavissa keskenään. Tämä tarkoittaa, että kaikki tuetut kielet pystyvät käyttämään koodia, joka on kirjoitettu millä tahansa muulla tuetulla kielellä. .NET Framework huolehtii suurimmasta osasta ohjelmien vaatimista välttämättömyyksistä, kuten esimerkiksi muistinhallinta, virheiden käsittely ja turvallisuus. Näin käyttäjä pystyy keskittymään oleelliseen eli ohjelmiston ns. business-logiikkaan. (Chauhan 2014; Microsoft 2016b)

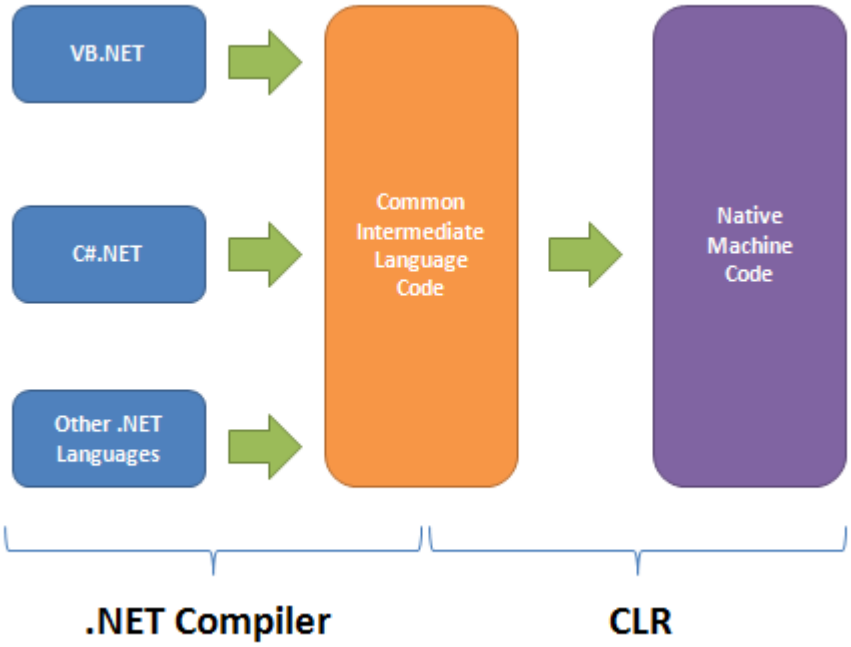
.NET Framework koostuu kahdesta osasta (Kuva 1). Näistä ensimmäinen on sen mittava luokkakirjasto eli Framework Class Library (FCL), joka pitää sisällään kattavan tarjonnan yleisiä toimintoja kuten tiedostojen kirjoittaminen sekä lukeminen, graafinen renderöinti ja vuorovaikutus tietokantojen ja kehitettävien ohjelmien välillä. Luokkakirjaston koko sisältö on saatavilla kaikilla .NET tuetuilla ohjelmointikielillä. Toinen osa on .NET ajonaikainenympäristö eli Common Language Runtime (CLR).

Kaikki .NET ympäristölle kehitetyt ohjelmat ohjelmointikielestä riippumatta suoritetaan CLR:n kautta. (Chauhan 2014; Microsoft 2016b)



Kuva 1. .NET Framework 4.5 rakenne. (Chauhan 2014)

CLR käyttää Just-In-Time (JIT) kääntötekniikkaa, jossa ohjelma käännetään kohdekielelle vasta ajon aikana. Ohjelmoijan kirjoittama koodi esikäännetään kehitysympäristössä välikielelle, jota kutsutaan nimellä Microsoft Intermediate Language (MSIL). Ajonaikaisen ympäristön tarjoama virtuaalikone ottaa tämän välikielisen koodin ja tulkitsee sitä edelleen käyttöjärjestelmän ymmärtämään binäärimuotoon ohjelmaa suoritettaessa (Kuva 2). Tämän tekniikan ansiosta sama kerran kirjoitettu koodi toimii erilaisilla suoritintyypeillä kuten myös tulevilla suorittimilla. Lopullinen konekoodi (native machine code) on tietylle prosessorityypille tarkoitettua binäärikoodia. (Bercero 2009; Chauhan 2014; Microsoft 2016b)



Kuva 2. Prosessi .NET tukemien kielten koodista ajettavaksi konekoodiksi. (Bercero 2009)

2.3 C# ohjelmointikieli

Projektissa käytettiin ohjelmointikielenä Microsoftin kehittämää C#-kieltä, joka sisältyy Microsoftin laajempaan .NET Framework ohjelmistokomponenttikirjastoon. C# on moderni, yleiskäyttöinen, vahvasti tyypitetty ja oliopohjainen ohjelmointikieli. Kieli kehitettiin vanhojen C-kielten pohjalta ja perii niiltä syntaksia ja rakenteita. Yksi kehityksen lähtökohtia oli, että kieli tuntuisi tutulta kenelle tahansa, joka tuntee aikaisempia C.kieliä. C-kieliin verrattuna se on helpompi omaksua ja hoitaa joitain vaikeammin ymmärrettäviä tehtäviä automaattisesti käyttäjän puolesta. Esimerkkinä tästä voidaan mainita ”roskien keruu” eli muistinhallintatehtävät. C#-kielessä käyttäjän ei tarvitse itse kirjoittaa tuhoamisrutiineja olioilleen käytön jälkeen. Roskienkerääjä aika ajoin automaattisesti siivoaa pois olioita, joihin ei enää viitata missään. (Ecma International 2006, xix)

C#-kielellä kehitetään pääasiassa Windows sovelluksia, mutta konsolipohjaisten sovellusten kirjoittaminen on myös mahdollista. Windows käyttöliittymien luomista varten kielelle löytyy Windows Forms, joka on .NET ympäristöön sisältyvä luokka-

kirjasto, jonka tehtävä on tarjota kattava työkaluvalikoima graafisten käyttöliittymien toteuttamista varten. Windows Formsiin perustuvat ohjelmistot ovat tapahtumapohjaisia ohjelmistoja tarkoittaen, että suuren osan ajasta ne yksinkertaisesti odottavat käyttäjän tekevän jotain, mistä aiheutuu tapahtuma. Tapahtuma voi olla vaikka napin klikkaus, joka laukaisee nappiin sidotun tapahtuman, jonka seurauksena sen koodi suoritetaan. (Microsoft 2016b) Kaikki Topologyn käyttöliittymät toteutettiin Windows Formsia käyttäen.

2.3.1 Tyypit

C# on vahvasti tyypitetty kieli, mikä tarkoittaa, että kaikilla objekteilla on tyyppi, joka on tiedossa ja saatavilla. Muuttujan tyyppiä ei voida muuttaa sen julistamisen jälkeen. Muuttujaan voidaan asettaa ainoastaan muuttujan tyyppisiä arvoja. Referenssityyppien tapauksessa kantaluokasta periytettyjen lapsiluokkien objekteja voidaan varastoida kantaluokan tyyppisiin muuttujiin. (Microsoft 2016b)

C#-kielessä on kahdenlaisia tyyppejä. Nämä tyyppiluokat ovat arvotyypit ja referenssityypit. Arvotyyppiset muuttujat yksinkertaisesti ja suoraan sisältävät niiden alta löytyvän tiedon, siinä missä referenssityypit varastoivat viittauksia (referenssejä) objekteihin. Referenssityypeissä kaksi tai useampikin muuttuja voi viitata samaan objektiin, jolloin yhtä muuttamalla pystytään vaikuttamaan kaikkiin samaan tietoon viittaaviin muuttujiin. Arvotyypin muuttujat sisältävät aina oman kopionsa tiedosta ja täten arvotyypin muuttujaan kohdistuvat operaatiot eivät voi vaikuttaa muihin arvotyypin muuttujiin. (Ecma International 2006, 16; Microsoft 2016b)

2.3.2 LINQ

Language Integrated Query (LINQ) on .NET luokkakirjastoihin sisältyvä komponentti, joka mahdollistaa SQL:n kaltaiset datakyselyt moniin erityyppisiin tietokokoelmiin, kuten esimerkiksi taulukoihin, XML dokumentteihin ja listoihin, suoraan C# syntaksilla. LINQ lausekkeet ovat hyvä vaihtoehto C# kielessä tyypillisesti käytetyille toistorakenteille. LINQ lausekkeilla toteutetut haut ovat koodillisesti toistora-

kenteilla toteutettuja hakuja huomattavasti luettavampia ja tiiviimpiä. (Kuva 3). (Microsoft 2016b) Topology projektissa LINQ lausekkeita hyödynnettiin lähes jokaisessa toteutetussa ohjelman osiossa.

```
private Element GetElementWithLinq(int elementID)
{
    return ElementList.SingleOrDefault(element => element.Elementti_ID == elementID);
}

private Element GetElementWithLoop(int elementID)
{
    for(int i = 0; i < ElementList.Count; i++)
    {
        if(ElementList[i].Elementti_ID == elementID)
        {
            return ElementList[i];
        }
    }

    return null;
}
```

Kuva 3. Kaksi metodia, joiden toiminta on täysin identtistä keskenään. Ensimmäinen on toteutettu LINQ lausekkeella ja jälkimmäinen perinteisellä toistorakenteella.

2.4 Microsoft Access

Tietokantojen hallinnointiin Topologyn käyttämässä tietokantaympäristössä käytetään yksinomaan Microsoftin Office-tuoteperheeseen sisältyvää Access ohjelmaa. Access käyttää tiedon tallentamiseen omaa formaattiansa, joka perustuu Jet-tietokantamoottoriin. Tietokantayhteydet Topologyn ja Access tietokantojen välillä toteutettiin OLE DB objektien välityksellä. OLE DB on Microsoftin kehittämä API (ohjelmointirajapinta), joka mahdollistaa tietojen lukemiseen ja kirjoittamisen tarvittavat yhteydet Access tietokantatiedostoihin. (Microsoft 2016b) Accessia projektiin liittyen käytetään lähinnä tietokantojen rakenteiden muokkaamiseen, eli uusien taulujen lisäämiseen ja niiden muokkaamiseen, kun taas tietojen syöttöön, lukemiseen ja muokkaamiseen on kehitetty omat ohjelmansa, joista Topology on yksi.

2.5 SQL kieli

Structured Query Language (SQL) on IBM:n kehittämä standardoitu kyselykieli, jonka avulla relaatiotietokantoihin voidaan tehdä hakuja, muutoksia ja lisäyksiä. Tärkeimmät SQL-kielen käskyt ovat SELECT, UPDATE, INSERT ja DELETE. Topology projektin yhteydessä SQL-kielen käyttö rajoittui hyvin pitkälti näihin peruskäskyihin. Lähestymistapana käytettiin yleisesti datan lataamista ohjelmaan SQL kyselyn avulla, jonka jälkeen sitä suodatettiin ja käsiteltiin ohjelman sisäisesti ennen muutosten tallentamista tietokantaan jälleen SQL päivityskomennolla. (IBM 2016)

2.5.1 Tietueiden haku

Tietueiden haku SQL-kielisesti tapahtuu SELECT lauseen avulla. SELECT lause koostuu kahdesta osasta. SELECT osassa määritellään mitkä kolumnit halutaan tietokannasta hakea ja FROM osassa määritellään mistä taulusta haluttavat tiedot haetaan (Kuva 4). Tämän jälkeen on mahdollista mm. suodattaa tietoa edelleen lisäämällä WHERE osio, jossa voidaan määritellä ehtolause. Tällöin kysely palauttaa ainoastaan ne tietueet, jotka täyttävät WHERE osion asettaman ehdon. Kolumnien nimet voidaan niin halutessa korvata tähdellä (*), jolloin kysely palauttaa kaiken tiedon kyseessä olevasta taulusta. (IBM 2016; W3Schools 2016)

```
SELECT Elementti_ID, Solmu_I, Solmu_J
FROM Elementit;
```

Kuva 4. Yksinkertainen SQL-kielinen tietuehaku, joka palauttaa elementtien solmutiedot ”Elementit”-taulusta.

2.5.2 Tietueiden päivitys

Tietueiden päivitys hoidetaan UPDATE lauseella. UPDATE lause koostuu tyypillisesti kolmesta osasta. UPDATE osassa määritellään mitä tietokannan taulua halutaan päivittää, SET osassa määritellään päivitettävät kolumnit sekä niiden uudet arvot ja lopulta WHERE osassa määritetään ehtolause, jonka perusteella määräytyy mikä tai

mitkä tietueet päivitetään (Kuva 5). WHERE lausekkeen voi tässäkin tapauksessa jättää kokonaan pois, mutta näin tehdessä päivitykset tehdään kaikkiin taulusta löytyviin tietueisiin. (IBM 2016; W3Schools 2016)

```
UPDATE Elementit
SET Tarkastusluokka=2
WHERE Erikoiselementtityyppi='Palkki';
```

Kuva 5. Yksinkertainen SQL-kielinen päivityslause, joka asettaa Elementit-taulussa Tarkastusluokka kolumnin arvoksi 2 kaikissa tietueissa, joiden Erikoiselementtityyppi on Palkki.

2.5.3 Uusien tietueiden lisäys

Tietueiden lisäys tietokantaan tapahtuu INSERT lauseella. Lause koostuu kahdesta pakollisesta ja yhdestä ehdollisesta osasta, jonka sisällyttäminen muuttaa lauseen toimintaa kriittisellä tavalla. INSERT INTO osassa määritetään mihin tauluun lisäyksiä ollaan tekemässä. Tämän jälkeen seuraa ehdollinen osa lauseesta, jossa voidaan määritellä mihin taulun kolumneihin tietoja ollaan lisäämässä. Jos tämä osa jätetään lauseesta pois, asettaa lause annetut arvot järjestyksessä taulun kolumneihin. VALUES osassa määritetään syötettävät arvot (Kuva 6). Jos täytettävät kolumnit määriteltiin ehdollisessa osassa, täytyy arvoja syöttää yhtä monta kuin määriteltyjä kolumneja, muuten lisäys epäonnistuu. (IBM 2016; W3Schools 2016)

```
INSERT INTO Elementit (Elementtinumero, Solmu_I, Solmu_J)
VALUES (1, 1001, 1002);
```

Kuva 6. SQL-kielinen lause, jolla lisätään Elementit taulun kolumneihin Elementtinumero, Solmu_I ja Solmu_J arvot 1, 1001 ja 1002 samassa järjestyksessä.

2.5.4 Tietueiden poistaminen

Tietueiden poistaminen tapahtuu DELETE lauseella. Lause koostuu kahdesta osasta. DELETE FROM osassa määritetään mistä taulusta poisto halutaan tehdä ja WHERE

osassa määritellään ehto, jonka täyttävät tiedot tullaan poistamaan (Kuva 7). WHERE osa on mahdollista jättää määrittelemättä, mutta näin tehdään hyvin harvoissa tapauksissa, sillä näin tehdessä lause poistaa kaikki tietueet kyseisestä taulusta. (IBM 2016; W3Schools 2016)

```
DELETE FROM Elementit
WHERE Elementti_ID=13;
```

Kuva 7. SQL-kielinen poistolause, joka poistaa Elementit taulusta tietueen, jonka Elementti_ID arvo on 13.

2.6 DevDept Eyeshot

Eyeshot on DevDept Softwaren kehittämä 3D grafiikka ja CAD-komponentti, joka toimii WinForms ympäristössä. Komponenttiin sisältyy lukuisia eri entiteettityyppejä, joiden avulla dataa voidaan mallintaa ruudulle kolmiulotteisina objekteina, sekä hyödyllisiä luokkia, joita voidaan hyödyntää kolmiulotteisen tilan käsittelyyn liittyvissä laskentatoimituksissa. Luokista mainittakoon esimerkkinä Vector3D, jonka alta löytyy staattisia metodeja mm. vektorien välisten risti- sekä pistetulojen laskentaan ja kolmiulotteisten pisteiden välisten suuntevektorien laskentaan. (DevDept Software 2016) Topologyssä Eyeshottia käytetään mallintamaan putkistoja tietokantaan tallennetun tiedon pohjalta.

2.7 Visual Basic 6.0

Alkuperäinen Topology-ohjelma on kirjoitettu Visual Basic kielellä, joka on Microsoftin kehittämä BASIC-sukuinen yleiskäyttöinen ohjelmointikieli. Kehitysympäristönä käytettiin Visual Basic 6.0, jonka tuen Microsoft lopetti vuonna 2008 (Microsoft 2016c). Näin ollen C# versiota kehittäessä oleellinen osa työtä oli Visual Basic kielisen koodin tulkitseminen ja toteuttaminen C#:lla hyväksikäyttäen joko samoja tekniikkoja tai aina kun mahdollista tehokkaampia moderneja tekniikkoja.

2.8 Ohjelmistotestaus

Ohjelmistotestauksella tarkoitetaan prosessia, jossa etsitään eroavaisuuksia ohjelmiston tai sen osan toiminnan ja odotetulle toiminnallisuudelle asetettujen vaatimusten välillä. Testausta harjoitetaan koko kehitysprosessin ajan ohjelmoijien toimesta ja tyypillisesti myöhemmässä vaiheessa kehitystä ohjelman käyttäjien tai testausta ammatikseen harjoittavien henkilöiden toimesta. Mitä aikaisemmassa vaiheessa virheet ohjelmakoodissa huomataan, sitä vähäisemmiksi niiden vaikutukset jäävät. (Williams 2006b, 34)

2.8.1 White box-testaus

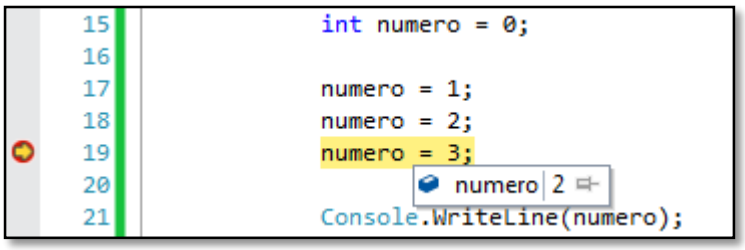
White box-testauksella tarkoitetaan ohjelmistotestausta, jossa testaaja on tietoinen ohjelmiston sisäisestä toiminnasta ja suunnittelee testitapaukset nimenomaan testaamaan tiettyä haaraa ohjelmiston koodista. Tämän tyyppisissä testeissä testaaja tietää syötettävien arvojen perusteella jo ennen testin suorittamista mitä ohjelman tulisi tuottaa. White box testauksesta käytetään myös termiä lasilaatikkotestaus, jolla viitataan ohjelmistorakenteen läpinäkyvyyteen testaajan näkökulmasta. Testaaja on tyypillisesti ohjelman ohjelmoija itse. (Williams 2006a, 60)

2.8.2 Black box-testaus

Black box-testauksella tarkoitetaan ohjelmistotestausta, jossa testaaja ei ole tietoinen ohjelmiston sisäisestä toiminnasta ja keskittyy ainoastaan määrittelemään täyttääkö ohjelma sille asetetut funktionaaliset vaatimukset. Black box-testaus on White box-testausta pinnallisempaa ja sen perusteella voidaan todeta ohjelman ”näyttävän” toimivan kuten kuuluukin. On oleellista, ettei testaaja ole ohjelman ohjelmoija itse, eikä tiedä ohjelmakoodin toteutuksesta mitään. Ohjelmakoodin itse kirjoittaneet henkilöt tapaavat alitajuisesti testata, että ohjelma tekee juuri sen mitä he ohjelmoivat sen tekemään eikä niinkään, että ohjelma tekee mitä sen kuuluisi sille asetettujen vaatimusten mukaan tekevän. (Williams 2006b, 43)

2.8.3 Pysäytyspistetekniikka

Pysäytyspistetekniikka (breakpoint) on ohjelmointiympäristöihin yleisesti kuuluva toiminto, joka löytyy myös Visual Studion kaikista eri versioista. Toiminto mahdollistaa ohjelmakoodin ajamisen rivi kerrallaan ohjelmointiympäristön sisäisesti. Näin koodin toimintaa voidaan tarkastella hyvin tarkasti tarpeen niin vaatiessa. Pysäytyspiste voidaan asettaa koodiin mille tahansa riville, jonka jälkeen ohjelmaa ajettaessa ohjelmointiympäristö tauottaa ohjelman ajon kyseiselle riville astuttaessa. Pysäytyksen aikana muuttujien arvoja voidaan tarkastella yksityiskohtaisesti (Kuva 8). Käyttäjä voi halutessaan jatkaa ohjelman suoritusta normaalisti tai astua eteenpäin rivi kerrallaan tarkastellen muutoksia muuttujissa. (Microsoft 2016b)



Kuva 8. Breakpoint asetettuna riville 19 keskeyttää ohjelman ajon riville astuttaessa, jolloin muuttujien sen hetkisiä arvoja pystytään tarkastelemaan.

2.9 Vektorilaskenta

Vektori on matemaattinen malli asialle, jolla on suunta ja suuruus. Vektorin alkioiden määrä määrittää vektorin ulottuvuuden. Tämän projektin yhteydessä vektoreista puhuttaessa puhutaan lähes yksinomaan kolmialkioisista (X, Y ja Z) eli kolmiulotteisista vektoreista. Vektoreita käytettiin lähinnä suuntien ilmaisemiseen. Globaalivektoreilla tarkoitetaan kaikkien kolmiulotteisten entiteettien jakaman tilan X, Y ja Z suuntia. Lokaalivektorit puolestaan määritellään yksittäisille entiteeteille tarpeen niin vaatiessa. Näin tehdään esimerkiksi tuille, kun ne halutaan piirtää globaaleista suunnista poikkeavaan asentoon. (Ohjelmointiputka 2007)

Yleisimmin käytetty laskutoimitus oli ristitulo, joka on käytännössä kolmiulotteisten vektorien välinen kertolasku. Tuloksena siitä saadaan vektori, joka on kohtisuorassa

molempia vektoreita vastaan. Näin pystytään esimerkiksi ratkaisemaan suuntavektori Z, kun tiedetään vektorit X ja Y. Toinen suhteellisen yleinen toimitus oli pistetulo, jonka tuloksena saadaan tavallinen skalaariluku. Tästä oli erityisen paljon hyötyä suuntavektoreita määritellessä, kun piti selvittää ovatko vektorit 90 asteen kulmassa toisiinsa nähden, kuten niiden kuuluisi olla. Vektorit ovat aina kohtisuorassa toisiaan vasten pistetulon ollessa 0. (Ohjelmointiputka 2007)

3 YMPÄRISTÖ

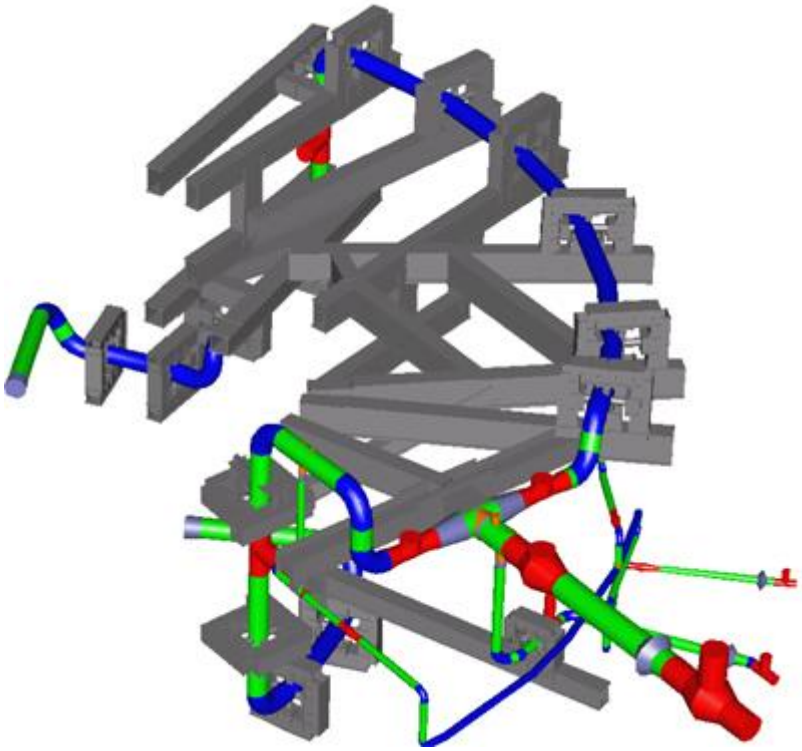
3.1 PAMS

PAMS on lyhenne sanoista Piping and Component Analysis and Monitoring System eli putkistojen ja komponenttien analysointi- ja valvontajärjestelmä. PAMS pitää sisällään lukuisia tietokantoja ja itsenäisiä ohjelmistokokonaisuuksia, joita käytetään Olkiluodon laitosyksiköiden turvallisen toiminnan varmistamiseksi. Topology on yksi näistä ohjelmistoista. Eri ohjelmistot käyttävät samoja tietokantoja ja niihin on rakennettu käyttöliittymiä, joiden välityksellä ohjelmat voivat vaihtaa tietoa keskenään. Ohjelmistoilla suoritetaan laitosyksiköiden kuormituksiin, lujuuteen ja tarkastuksiin liittyviä analyysejä. PAMS kehitys alkoi vuonna 1995. Tietomäärien vuoksi tiedon varastointi ja hakeminen fyysisestä paperiarkistoinnista oli muodostunut haasteelliseksi tehtäväksi. Tietojen hallintaan ja etsimiseen täytyi löytää parempi ratkaisu. PAMS kehitys on edelleenkin jatkuva tehtävä. (TVO 2013)

3.2 Topology ja putkistotietokanta

Topology ja sen käsittelemät tietokannat muodostavat keskeisen osan PAMS:sta. Tietokannat sisältävät tietoa mm. laitosyksiköiden putkistojen rakenteista ja kuormituksista. Topologyä käytetään näiden tietojen lukemiseen ja kirjoittamiseen sekä tietojen pohjalta tehtävään kolmiulotteiseen mallinnukseen kuten kuvassa 9. Visuaalisen luonteensa vuoksi Topology soveltuu erinomaisesti myös rakenteiden demon-

stroimiseen. Kolmiulotteista kuvaa katselemalla tietokannasta on myös helppo löytää virheellisiä tietoja kuten epäsymmetrisiä rakenteita.



Kuva 9. Topologyllä mallinnettu syöttövesiputkisto ja sen tukirakenteet. (TVO 2013)

4 KUVAUS TYÖN ETENEMISESTÄ

4.1 Tilannekartoitus

Topology C# projektia oli ajettu eteenpäin lähinnä kesä- ja opinnäytetöiden muodossa, joten ohjelman parissa työskennelleitä ohjelmoijia oli lukuisia. Ohjelmoijien myötä erilaisia toteutustapoja ohjelmakoodiin oli kertynyt lukuisia ja yhtenäisen toteutustavan ja dokumentoinnin puuttuessa projektin valmiusaste oli hämärtynyt ajan myötä. Työn ensimmäinen vaihe oli rakentaa lista kaikista Topology-ohjelman osista ja testata niiden toiminnallisuus aiempaa ohjelman versiota vasten, jotta saatiin yleisluontoinen kuva projektin tilasta.

Ohjelman osat luokiteltiin seuraavan luokitusjärjestelmän mukaisesti:

- 0: Toteuttamaton
- 1: Osittain toteutettu, mutta ei luotettava
- 2: Osittain toteutettu ja luotettava
- 3: Täysin toteutettu, mutta ei luotettava
- 4: Täysin toteutettu ja luotettava

Testaus suoritettiin ensin Black box-testauksena, jonka perusteella päätettiin oliko White box-testaukselle tarvetta. Ohjelman osien toiminnan taso oli määriteltävissä pelkällä Black box-testauksella tapauksissa, joissa osia oli täysin vailla toteutusta tai ne ilmiselvästi toimivat väärällä periaatteella verratessa Topologyn VB versioon.

Haastavampia testattavia olivat osat, joissa pinnallisesti tarkasteltuna osa näytti toimivan kuten kuuluukin, mutta tarkemmin tarkasteltuna kaikki ohjelmakoodin haarat eivät olleet luotettavia. Testitapaukset luotiin tulkitsemalla aiemman ohjelman VB koodia ja sen perusteella päättelemällä mitä ohjelma tuottaa tulokseksi milläkin syötteillä. Yksityiskohtaiseen koodin tutkimiseen ja eri haarojen testaukseen käytettiin pysäytyspistetekniikkaa Visual Studiossa.

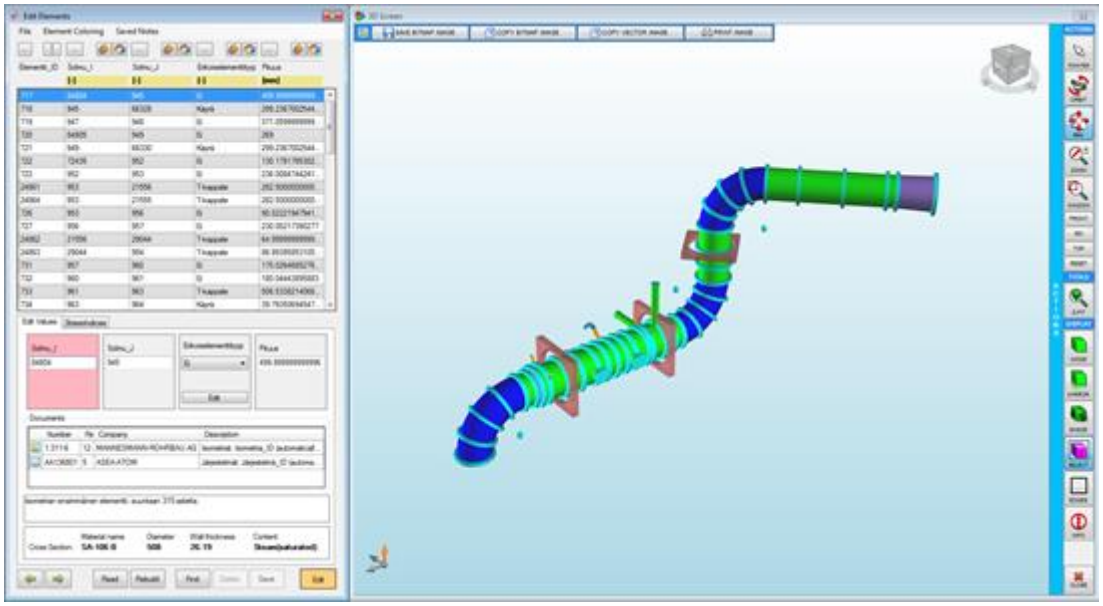
Luokituksen jälkeen listaa jalostettiin edelleen määrittelemällä jokaiselle osalle erikseen tultaisiinko osaa ylipäänsä toteuttamaan ja prioriteettiarvo. Prioriteettiarvoja oli kaksi, joista ensimmäiseen ryhmään kuuluvia komponentteja käytetään viikoittaisella tasolla ja jälkimmäiseen kuuluvia harvemmin. Tässä yhteydessä sovittiin, että ohjelma olisi valmis laajempaan käyttäjätestaukseen, kun kaikki ensimmäisen prioriteetin omaavat komponentit olisivat täysin ja luotettavasti toteutettuja.

Tilannekartoituksen perusteella kävi ilmi, ettei ohjelman kaikkien komponenttien saattaminen beta-testausasteelle ollut realistinen tavoite käytettävissä olleessa ajassa. Toteutettavaksi rajattiin solmujen ja elementtien editointiin tarkoitetut työkalut, joiden toteutuksissa tilannekartoituksen aikana havaittiin niin suuria puutteita, että komponentit oli tarve toteuttaa kokonaan uudestaan. Näiden komponenttien toteutuksessa oli määrä käyttää keskenään yhtenäistä toteutustapaa, joten ne yhdessä muodostivat sopivan kokonaisuuden.

4.2 Elementtien ja solmujen editointityökalut

Putkistojen rakenteita kuvaavien tietokantojen perustavina rakennuspalikoina toimivat elementit, niiden solmut ja solmuihin liitetyt tuet. Elementti kuvaa fyysistä osaa putkistosta ja se voi olla tyypiltään esimerkiksi suoraputki tai palkki. Elementteihin on aina liitetty vähintään kaksi solmua. Nämä ovat elementin alku ja päätepisteet. Niitä kutsutaan nimillä I ja J solmut. Jos elementin tyyppi niin vaatii, on elementtiin liitetty myös apusolmuja, joita tarvitaan avustamaan elementtien piirtoa. Esimerkiksi kaarevan putken tapauksessa elementeiltä löytyy ympyrän keskipisteen määrittelevä apusolmu, jonka ympäri putki kaareutuu.

Elementtien ja solmujen editointityökaluissa havaittiin tilannekartoituksessa niin suuria puutteita, että nämä ohjelman osat koettiin parhaaksi toteuttaa uudestaan tyhjältä pöydältä yhtenäisenä kokonaisuutena niin pitkälti kuin mahdollista. Molemmat työkalut käyttävät samanlaista käyttöliittymää, jonka keskeisimpänä osana on Windows Formsin DataGridView-tyyppinen taulukkoelementti (Kuva 10). Käyttäjän käynnistäessä työkalu, se lataa automaattisesti kaikkien 3D-ikkunassa piirrettynä olevien elementtien tai solmujen tiedot taulukkoon.



Kuva 10. Elementtien editointityökalun käyttöliittymä.

Taulukko ja 3D-ikkuna sidottiin toisiinsa siten, että käyttäjän valitessa taulukosta rivi tai rivejä, 3D-ikkuna maalaa kuvassaan valittuna olevia rivejä vastaavat objektit normaalista poikkeavalla värillä. Sama toimii myös toisinpäin, eli 3D-ikkunasta voi valita objektin sitä klikkaamalla ja mikäli objektia vastaava rivi löytyy taulusta, se valitaan automaattisesti. Näin käyttäjä pystyy helposti paikantamaan haluamansa objektin rakenteesta.

Turvarakenteena molempiin työkaluihin rakennettiin eritellyt moodit tietojen lukemiseen ja muokkaamiseen. Lukumoodissa suuri osa työkalun käyttöliittymästä on lukittu pois käytöstä. Tällä pyrittiin estämään tietojen tahatonta muokkausta. Käyttäjän halutessa muokata tietoja, täytyy hänen astua muokkausmoodiin manuaalisesti klikkaamalla siihen tarkoitettua nappia. Tämä avaa käyttöliittymän lukitut osat käyttöä varten.

4.2.1 Esitettävien arvojen valinta

Elementti- ja solmutaulukoissa esitettävät tiedot ovat käyttäjän itse valittavissa. Ensimmäisessä sarakkeessa esitetään aina elementin tai solmun ID tai numero. Muissa sarakkeissa voidaan esittää mikä tahansa tietokannasta löytyvä elementtiin tai solmuun liitetty tieto.

Valinta tapahtuu klikkaamalla sarakkeen yläpuolelta löytyvää nappia, jota painaessa ohjelma luo ContextMenu-tyyppisen valikon ja esittää sen käyttäjälle. Valikko päätettiin luoda tässä vaiheessa aina uudelleen, sillä osana toiminnallisuutta otsikko merkataan ruksilla, jos se on valittuna jo valmiiksi jossain muussa sarakkeessa. Kuvassa 11 C#-kielinen toistorakenne rakentaa tietokannasta ladattujen sarakenimien perusteella MenuItem-tyyppiset oliot ja lisää ne rakennettavaan ContextMenuun.

```
foreach (DataColumn dc in dt.Columns)
{
    // Skipattavat otsikot
    if (dc.ColumnName == "Solmu_ID" || dc.ColumnName == "Solmunumero" || dc.ColumnName == "Huomautukset")
        continue;

    // Luodaan uusi otsikko valikkoon
    menuItem = new MenuItem(dc.ColumnName);
    menuItem.Tag = "Solmut";
    menuItem.Click += mi_Click;
    if (titles.Contains(dc.ColumnName)) menuItem.Checked = true;
    titleMenu.MenuItems.Add(menuItem);
}
```

Kuva 11. C#-kielinen toistorakenne, jolla ContextMenu-valikon otsikot luodaan.

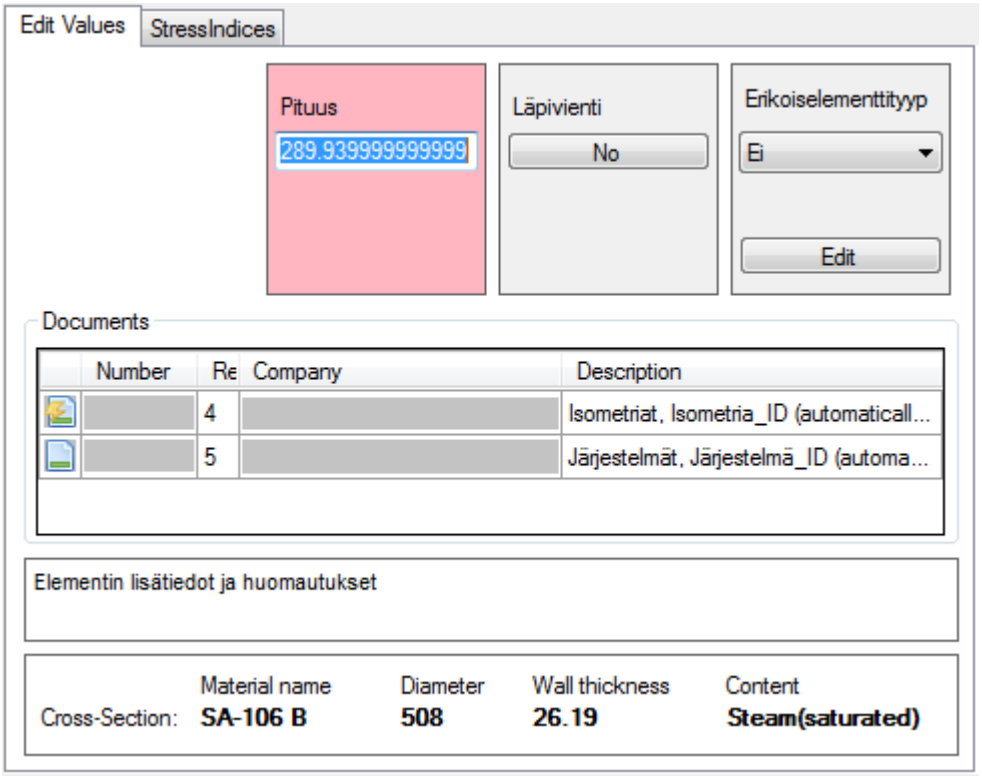
Tässä vaiheessa liitetään myös jokaisen MenuItemin tagiin tieto siitä, mihin tauluun tieto sisältyy, jotta käyttäjän klikatessa jotain näistä otsikoista ohjelma osaa etsiä tietoa oikeasta taulusta. Niissä tapauksissa, joissa sarake on viittaus johonkin muuhun tietokannan tauluun, luodaan alivalikko, johon jälleen listataan viitatun taulun sarakkeet. Alivalikot eivät muodostu automaattisesti vaan niitä varten ohjelmoitiin poikkeustapaukset. Ohjelman luodessa valikkoa jokaisen otsikon kohdalla tarkastetaan löytyykö kolumnin nimi poikkeustapausten merkkijono tyyppiseltä listalta. Jos nimi täsmää johonkin listasta löytyvään arvoon, on kyseessä alivalikollinen MenuItem, jonka alle luodaan oma valikkonsa samaa rutiinia käyttäen kuin aiemmin.

4.2.2 Perusarvojen editointi

Perusarvojen editointi molemmissa solmujen sekä elementtien editointityökaluissa toteutettiin samoja periaatteita käyttäen, jotta mahdollisten tulevien muutosten toteuttaminen kävisi mahdollisimman vaivattomasti kumpaankin työkaluun. Editointilaatikot ilmestyvät käyttöliittymässä esiin automaattisesti käyttäjän valitessa otsikon, jonka alta löytyvää tietoa halutaan editoida. Kaikkia tietoja ei voida näiden työkalujen kautta editoida ja esimerkiksi elementin solmutietoja tai solmujen koordinaatteja editoidakseen täytyy käyttäjällä olla peruskäyttäjää laajemmat käyttöoikeudet ohjelmassa.

Koska editointilaatikoiden tiedon varastointitarpeet olivat hyvin monimuotoisia tyypistä riippuen, toteutettiin tähän tarkoitukseen yksinkertainen EditBox-luokka. Luokka pitää sisällään muun muassa tiedot laatikon tyypistä, sen sisällään pitämästä ar-

vosta ja siitä, onko arvoa muutettu tietokantahaun jälkeen. Arvot luokkaan tallennetaan aina merkkijonotyyppisesti ja luokka pitää sisällään metodeja tähän tarkoitukseen tarvittaviin tyyppimuunnoksiin. Laatikot päivitetään välittömästi uuden otsikon valinnan nostattaman tapahtuman kautta.



Kuva 12. Editointilaatikot erityyppisine arvoineen. Ensimmäinen laatikko piilotettu, koska otsikkovalintaa ei ole tehty tai valitun otsikon arvo ei ole muokattavissa.

Editoitavia arvoja on kolmea tyyppiä: tekstiarvot, boolean arvot ja monivalintaiset arvoryhmät. Näistä kaikki ovat edustettuna kuvassa 12. Arvon ollessa monivalintaista tyyppiä ohjelma tarjoaa käyttäjälle automaattisesti alasvetovalikon, josta käyttäjä voi valita yhden sallituista arvoista. Boolean tyyppisissä tapauksissa käyttäjälle esitetään nappi, jota painamalla arvo muuttuu päinvastaiseksi. Eli jos arvo on alun perin tosi, nappia painamalla se voidaan muuttaa epätodeksi ja sama pätee toisinpäin. Tekstityyppiset arvot käyttäjä voi itse kirjoittaa esitettävään tekstilaatikkoon, mutta joidenkin tiettyjen arvojen yhteydessä tekstilaatikon lisäksi painettavissa on nappi, joka avaa tyypillisesti jonkun toisen ohjelman komponentin. Avatussa komponentissa voidaan esimerkiksi esittää listaus vaihtoehtoisista arvoista. Käyttäjän valitessa

arvon ohjelma sulkee avatun komponentin ja kirjoittaa tiedon automaattisesti tekstilaatikkoon.

Arvojen tallennukseen on kaksi tapaa. Käyttäjä voi painaa enter-näppäintä syötettyään haluamansa arvon laatikkoon tai klikata käyttöliittymästä löytyvää Save-nappia. Nappia pidetään lukittuna kunnes käyttäjä muuttaa mitä tahansa muokattavista arvoista. Muutosten nostattamat Changed-tapahtumat aktivoivat napin ja muuttavat sen taustavärin ilmoittaen käyttäjälle arvojen muuttuneen. Näin käyttäjä tietää aina onko esillä tallentamatonta tietoa. Tallennusrutiinin yhteydessä nappi lukitaan jälleen odottamaan uusia muutoksia. Tallennusrutiini tallentaa myös lisätiedot kentän, johon käyttäjä voi vapaasti kirjoittaa elementtiin liittyviä asioita (Kuva 12).

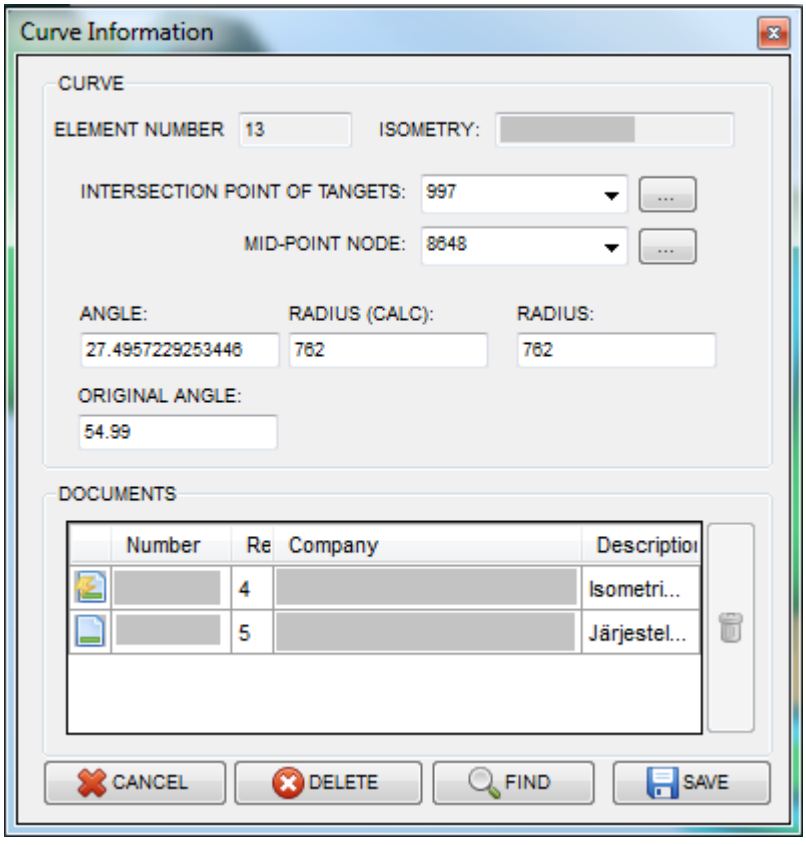
Tässä yhteydessä voidaan myös tehdä dokumenttiliitoksia otsikkoihin. Valittuna oleva laatikko korostetaan punaisella (Kuva 12) ja kyseisen laatikon otsikkoon liittyvät dokumentit ladataan dokumenteille tarkoitettuun listaukseen. Rivit ladataan dokumenttitietokannasta, jonka käsittelyyn on oma ohjelmansa nimeltä ReportBase. Käyttäjän painaessa Find-nappia, jonka kautta dokumentteja liitetään otsikoihin, käynnistetään ReportBase-ohjelma. ReportBase:n kautta käyttäjä voi selata dokumentteja ja valita haluamansa, jolloin ReportBase palauttaa Topologylle kyseisen dokumentin ID:n ja liitos luodaan tietokantaan. Jos käyttäjä kaksoisklikkaa dokumenttia käyttöliittymän listauksesta, avataan jälleen ReportBase-ohjelma, jonka kautta käyttäjä voi lukea kyseessä olevaa dokumenttia.

4.2.3 Erikoiselementtityyppikohtaiset editointityökalut

Jokaisella elementillä on tarkoitus olla erikoiselementtityyppi, mikä tarkoittaa, että elementille löytyy lisätietoja muista tietokannan tauluista. Jokaiselle erikoiselementtityypille on putkistotietokannassa oma taulunsa. Jos elementille on merkitty erikoiselementtityypiksi esimerkiksi ”Käyrä”, silloin voidaan olettaa, että käyrätaulusta löytyy samaa elementti ID:tä vastaava rivi. Rivi sisältää elementtiin liittyvää tietoa, jota saatetaan tarvita elementin piirtämiseen. Käyrän tapauksessa piirtäminen ei onnistu mikäli riviä ei löydy, sillä käyrää ei voida piirtää tietämättä sen apusolmujen sijaintia. Jos elementin erikoiselementtityyppi on ”Ei”, piirretään se suorana putkena.

Kun erikoiselementtityyppi valitaan otsikoksi, käyttäjälle tarjotaan mahdollisuus muokata olemassa olevan erikoiselementin tietoja, poistaa olemassa oleva erikoiselementti tai lisätä uusi erikoiselementti Edit-napin kautta (Kuva 12). Yhdellä elementillä voi samanaikaisesti olla ainoastaan yksi erikoiselementtityyppi. Jos käyttäjä yrittää vaihtaa erikoiselementtityyppiä alasvetovalikosta, tarkistaa ohjelma ensimmäiseksi onko elementillä jo ennestään olemassa erikoiselementtiä. Jos erikoiselementti löytyy, pyydetään käyttäjää poistamaan vanha erikoiselementti ennen uuden lisäämistä. Erikoiselementtityyppejä on yhteensä 16, joista jokaiselle toteutettiin oma editointityökalunsa. Kaikki toteutettiin samalla kaavalla niin pitkälti kuin erikoiselementtityypin tietokantarakenne niin salli.

Työkalu odottaa saavansa sitä kutsuvalta formilta lähtötietonaan editoitavan elementin ID:n. Jos ID:tä ei anneta tai ID on 0, olettaa työkalu kyseessä olevan uuden erikoiselementin lisäys. Tässä tapauksessa käyttöliittymään asetetaan oletusarvot valmiiksi. Jos taas ID saadaan, lataa työkalu valmiiksi editoitavan erikoiselementin tiedot niille tarkoitettuun käyttöliittymään (Kuva 13).



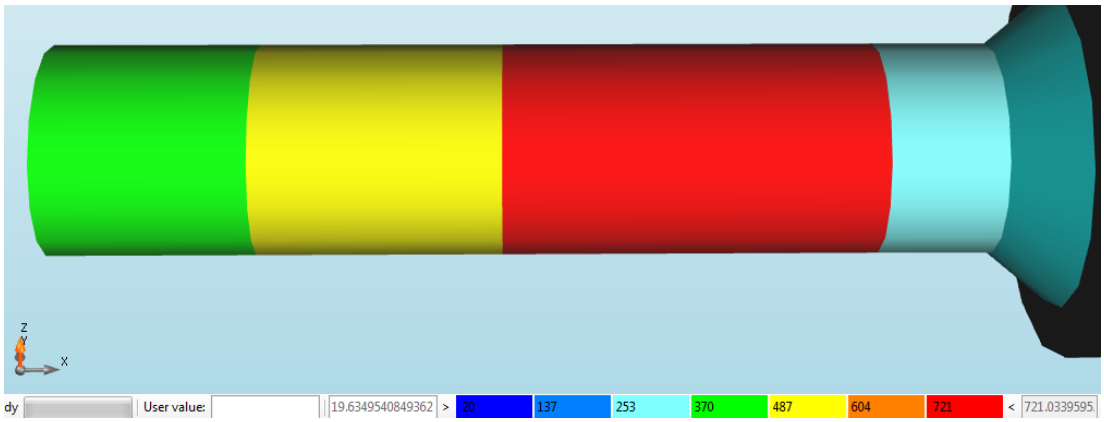
Kuva 13. Käyrän editointiin tarkoitetun työkalun käyttöliittymä.

Käyttöliittymän kautta käyttäjä pystyy muokkaamaan erikoiselementin tietoja. Dokumenttiliitosten tekeminen on myös mahdollista, mutta ainoastaan jos muokataan jo olemassa olevaa erikoiselementtiä. Liitosten tekeminen ei ole mahdollista ennen erikoiselementin tallentamista kantaan, sillä liitokset tehdään erikoiselementin ID:n kautta, jonka se saa vasta ensimmäisen tallennuksen yhteydessä. Jos kyseessä on jo olemassa oleva erikoiselementti, voidaan se poistaa Delete-napin kautta, jolloin rivi poistetaan tietokantataulusta ja elementin erikoiselementtityyppi palautetaan oletusarvoonsa, joka on ”Ei”, näin mahdollistaen uuden erikoiselementin lisäyksen. Save-nappia painettaessa ohjelma luo uuden erikoiselementin tai päivittää jo olemassa olevan erikoiselementin tiedot.

4.2.4 Arvovärityket

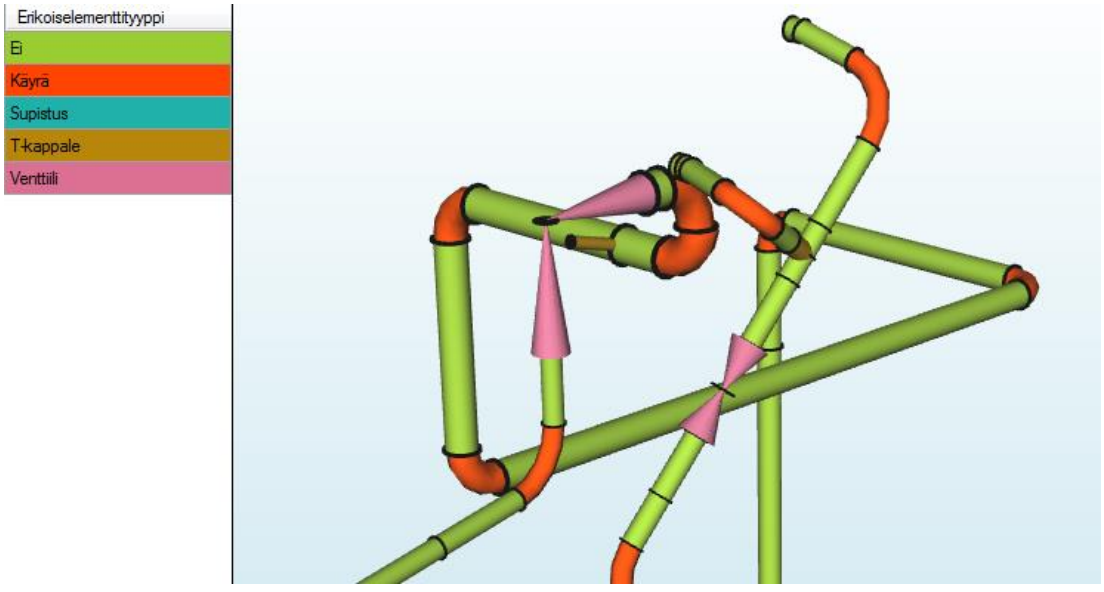
Molempiin työkaluihin toteutettiin arvoväritystoiminnot. Arvoväritykset tehdään otsikkokohtaisten arvojen perusteella ja ne voidaan suorittaa minkä tahansa arvojen perusteella muutamaa reunaehtoa kunnioittaen. Solmutyökalussa väritetään solmut ja elementtipuolella elementit. Väritykseen soveltumattomat objektit väritetään mustaksi. Arvovärityksiä on kahta eri tyyppiä.

Liukuvärityksessä arvot järjestetään pienimmästä suurimpaan. Suurimmalle arvolle asetetaan väriksi punainen ja pienimmälle sininen. Loput arvot suhteutetaan näiden arvojen välille ja väri valikoidaan dynaamisesti arvon perusteella. Näin voidaan helposti visuaalisen kuvan perusteella silmäillä arvoja ja saada yleiskuva arvojen välisistä eroista vierekkäisissä elementeissä. Jotta liukuväritys voidaan suorittaa, täytyy arvojen olla numeerisia ja pienin arvo ei saa olla yhtä suuri kuin suurin arvo. Kuvassa 14 elementit on liukuväritetty niiden pituuden perusteella.



Kuva 14. Liukuväritys elementtien pituuden perusteella.

Palettivärityksessä jokaiselle uniikille arvolle asetetaan oma värinsä ja kappale väritetään. Palettiväritys pystytään suorittamaan minkä tahansa tyyppisillä arvoilla, sillä kaikki arvot vertailuvaiheessa muunnetaan merkkijonotyyppisiksi. Käytännössä väritys onnistuu vaikka yhdellekään elementille ei olisi annettu arvoa, jonka perusteella väritys tehdään. Silloin kaikki kappaleet väritettäisiin mustaksi. Värityksen valmistuttua 3D-ikkunan reunassa esitetään taulu kaikista värityksessä käytetyistä arvoista ja niiden väreistä (Kuva 15).



Kuva 15. Erikoiselementtityypin perusteella tehty palettiväritys.

Palettivärityksen käynnistyessä luodaan lista kaikista taulun uniikeista arvoista. Tämän jälkeen luodaan KeyValuePair-tyyppinen lista, jossa Key-ominaisuutena toimii merkkijonotyyppinen arvo ja Value-ominaisuutena Color-tyyppinen väriarvo. Seu-

raavaksi käydään läpi aiemmin luotu lista uniikeista arvoista ja jokaista kohti luodaan KeyValuePair-olento, jossa Key-ominaisuuteen asetetaan taulun uniikki arvo ja Value-ominaisuuteen haetaan ennalta määritellystä värilistauksesta väri kyseessä olevan indeksin perusteella.

Näiden valmistelujen ollessa valmiit, voidaan käydä läpi kaikki 3D-entiteetit ja mikäli kyseessä on väritettävän tyyppinen entiteetti, asettaa sille väri etsimällä se entiteetistä löytyvän arvon perusteella aiemmin luodusta KeyValuePair-listauksesta. Jos kyseessä ei ole oikean tyyppinen entiteetti tai entiteetillä ei ole haluttua arvoa, väritetään se mustaksi. Kuvassa 16 on esitetty solmujen palettivärityksen väritysosuudesta huolehtiva toistorakenne. Elementtipuolella väritys toimii täysin samalla periaatteella.

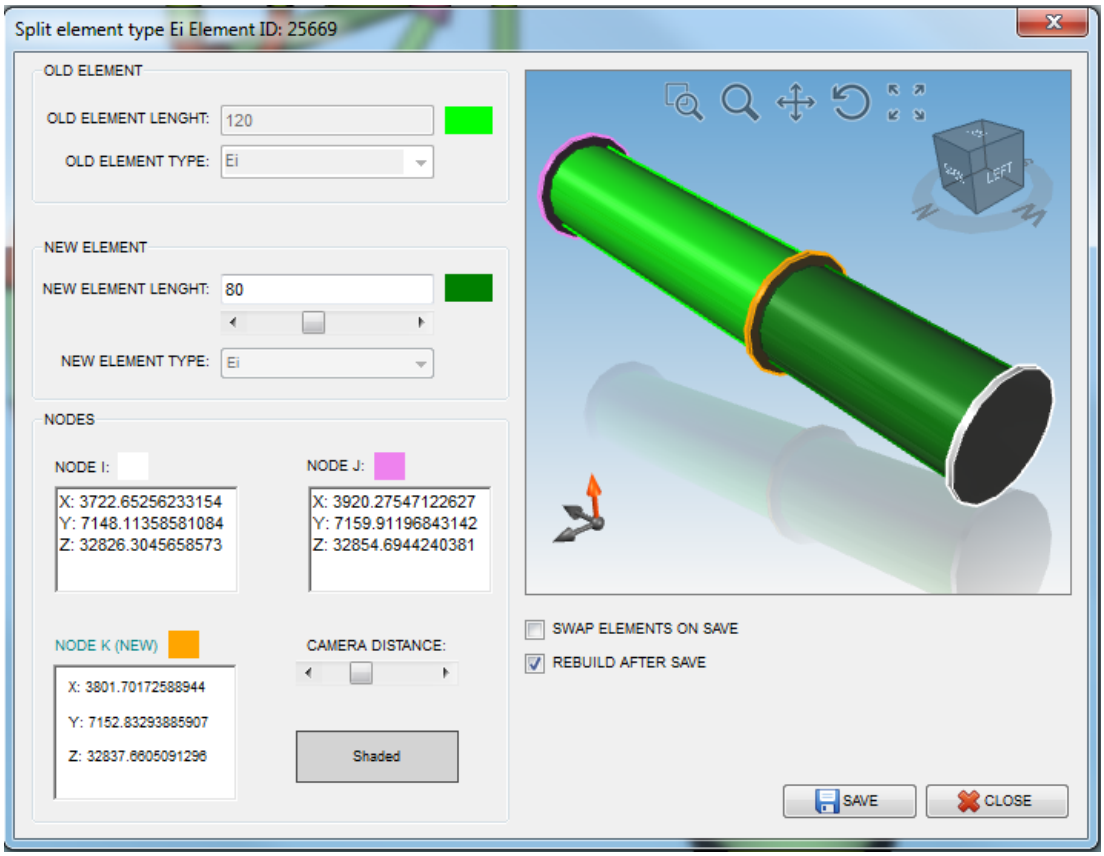
```
// Käydään kaikki 3D objektit läpi yksi kerrallaan
foreach (Entity ent in Program.Viewport3D.Entities)
{
    // Ilman entitydataa tyypin määrittäminen on mahdotonta, joten siirrytään seuraavaan
    if (ent.EntityData == null) continue;

    // Jos entiteetti kuuluu solmulle
    if (ent.EntityData is Node)
    {
        // Haetaan solmu gridistä käymällä rivit läpi
        foreach (DataGridViewRow dgvr in dgvNodes.Rows)
        {
            // Jos löytyi entiteetin Solmu_ID:tä vastaava rivi
            if (Convert.ToInt32(dgvr.Cells["sID"].Value) == ((Node)ent.EntityData).Solmu_ID)
            {
                // Jos solmulle on annettu väritettävä arvo
                if (!string.IsNullOrEmpty(dgvr.Cells[columnName].Value.ToString()))
                {
                    // Haetaan numeerista arvoa vastaava väri
                    luku = findColor(dgvr.Cells[columnName].Value.ToString());
                    sVari = cColorFunctions.SelectColor(luku);

                    // Tehdään väritys ja siirrytään seuraavaan objektiin
                    ent.Color = sVari;
                    break;
                }
                else { ent.Color = Color.Black; break; }
            }
        }
    } // Jos entiteetti kuuluu elementille, väritetään mustaksi asetuksen ollessa päällä
    else if (ent.EntityData is Element)
    { if (this.miElementsToBlack.Checked) { ent.Color = Color.Black; } }
}
```

Kuva 16. Solmujen palettivärityksen väritysosuudesta huolehtiva toistorakenne.

4.2.5 Elementin halkaisutyökalu

Elementin editointityökalun alle toteutettiin elementtien halkaisuun käytettävä työkalu. Työkalun tehtävä on lyhentää valittua elementtiä, luoda uusi elementti lyhennyskohdasta vanhan elementin päätesolmuun ja luoda lyhennyskohtaan uusi solmu, josta tulee lyhennetyn elementin uusi J-solmu ja uuden elementin I-solmu tai vaihtoehtoisesti sama toisinpäin. Työkalulla voidaan halkaista elementtejä joiden erikoiselementtityyppi on suoraputki tai palkki.

Uuden elementin pituuden voi määritellä joko syöttämällä sen numeerisesti sille tarkoitettuun tekstilaatikkoon tai vierittämällä tekstilaatikon alla olevaa vierityspalkkia, joka päivittää arvoa tekstilaatikkoon. Tekstilaatikon arvonmuutostapahtumaan on sidottu metodi, joka piirtää käyttöliittymän 3D-kuvan uudestaan. Näin kuva päivittyy reaaliajassa arvoa muuttaessa. Käyttäjä voi halutessaan muuttaa 3D-objektien värejä klikkaamalla niille tarkoitettuja värinappeja, jolloin esitetään Windows API:n tarjoama värinvalintaikkuna. Nappien värien muuttuessa väritykset päivitetään 3D-kuvaan automaattisesti ja välittömästi. Kuvassa 17 on esitetty työkalun käyttöliittymä.



Kuva 17. Elementin halkaisutyökalu.

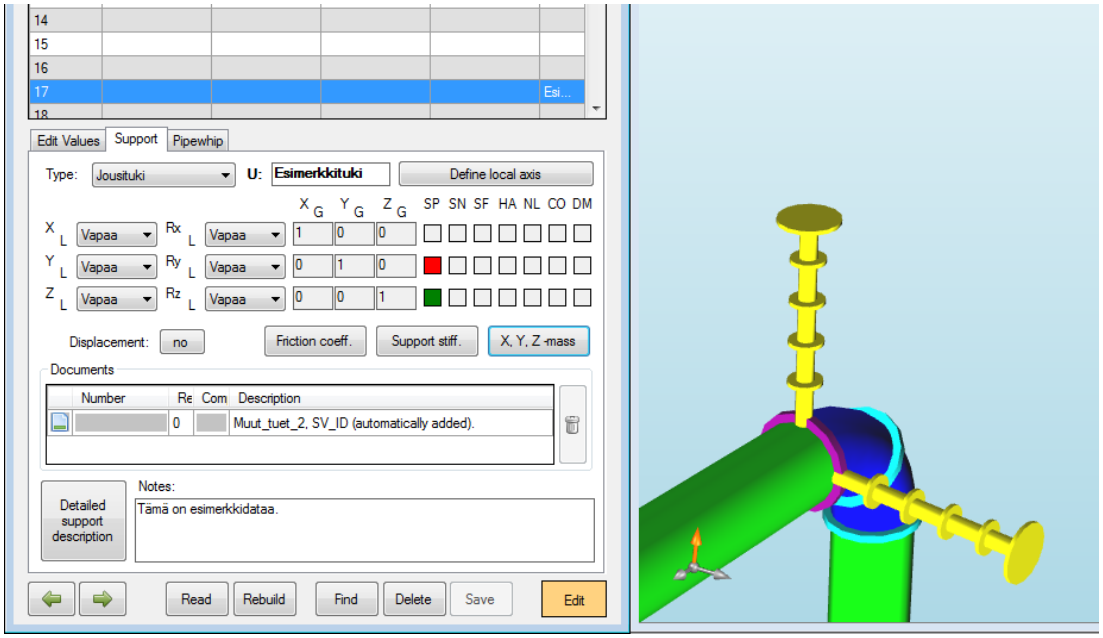
Muutokset kantaan tehdään painamalla Save-nappia. Uuden elementin perustiedot ja erikoiselementtityyppikohtaiset tiedot kopioidaan alkuperäisestä elementistä. Mikäli Swap-asetus on valittuna tallennusta tehdessä vaihtavat uusi ja vanha elementti paikkoja tallennuksen yhteydessä. Rebuild-asetuksen ollessa valittuna, haetaan ohjelman pää 3D-ikkunassa piirrettynä oleva alue uudelleen tietokannasta ja tehdään piirto uusien tietojen perusteella. Näin juuri lisätty elementti tulee välittömästi näkyviin 3D-entiteettinä tallennuksen jälkeen.

4.2.6 Tukityökalu

Tietokantarakenteessa tuet ovat aina liitettynä solmuun samoin kuin solmut ovat aina liitettynä elementteihin. Siksi solmutyökalun alle toteutettiin tukityökalu, jonka kautta pystytään lisäämään, poistamaan ja muokkaamaan solmujen tukia (Kuva 18). Tukityyppejä on yhteensä 16 ja rakenteen kannalta ajatellen nämä voidaan jakaa kahteen ryhmään. Ensimmäisen ryhmän tuet tallennetaan kantaan yleiseen tukitauluun ja

jälkimmäisen ryhmän tuet vaativat yleisen tukitaulun rivin lisäksi tietueen tai tietueita erillisissä tukityyppikohtaisissa tukitauluissa. Jälkimmäisen ryhmän tuet siis tavallaan liitetään tukeen ja yhdessä tuessa niitä voi olla rajaton määrä.

Jos työkaluun astuttaessa solmusta ei löydy tukea, on työkalun käyttöliittymä lukittu lukuun ottamatta tukityypin alasvetovalikkoa ja tuen koodikenttää. Kumman tahansa arvoa muuttamalla ja Save-nappia painamalla luodaan solmuun uusi tuki.



Kuva 18. Tukityökalun käyttöliittymä.

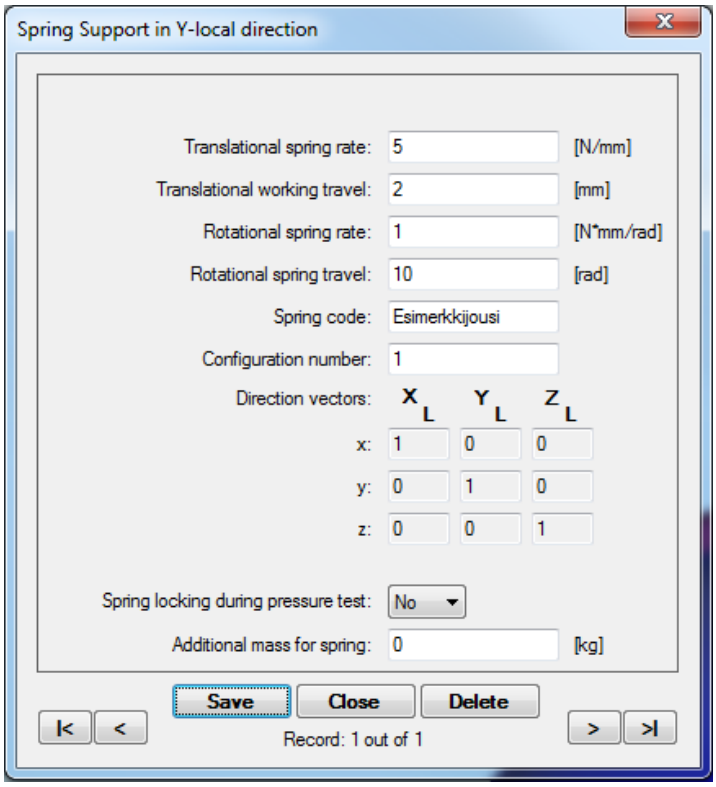
Yleisen tukitaulun tuet määritellään valitsemalla oikea tukityyppi ja asettamalla X, Y, tai Z suunnan tuenta ”Kiinni”. Tämä tarkoittaa, että elementti on tuettu kyseisellä tu-ella niin, ettei se pääse liikkumaan kyseisessä suunnassa. Tämän tyyppisiä tukia ovat muun muassa kannake- ja kiinnituet. Piirrot suoritetaan aina tukityypin mukaisesti, eli vaikka tuelle olisi määritelty jousityyppisiä alitukia, ei niitä piirretä, ellei tuen tyyppi ole jousituki.

Friction coeff- napin takaa aukeavassa formissa voidaan määritellä tuen kitkakertoimet, Support stiff-napin takaa määritellään tuen jäykkyys, XYZ mass-napista määritellään tuennan massakertoimet ja Displacement-napista määritellään siirtymät. Detailed support information-napista voidaan lukea ja kirjoittaa tuelle pidempi kommentti tukeen liittyen. Lisäksi työkalussa on Find-nappia painamalla mahdollisuus

tehdä dokumenttiliitoksia ReportBase-ohjelmaa hyväksi käyttäen. Dokumentit liitetään nimenomaan valittuna olevaan tukeen.

4.2.7 Tukityyppikohtaiset editointityökalut

Kuvassa 18 tukityökalun oikeassa reunassa näkyvät lukuisat napit ovat tarkoitettuja alitukien editointityökaluihin käsiksi pääsemiseen. Näitä alitukia on seitsemää tyyppiä, jotka ovat jousi-, snubber-, SFS, roikko-, epälineaarinen-, vakiovoima- ja vaimennintuet. Näitä kaikkia voi määritellä rajattoman määrän ja määrittelyt tehdään suuntakohtaisesti, eli X, Y tai Z lokaalivektorin suuntaisesti. Tietojen latausvaiheessa napit väritetään olemassa olevien tukien mukaisesti. Jos tukia ei löydy, nappi on harmaa, yhden tuen löytyessä nappi on punainen, kahden tuen löytyessä vihreä ja kolmen tai enemmän tuen tapauksessa nappi väritetään siniseksi.



Kuva 19. Jousitukien editointityökalun käyttöliittymä.

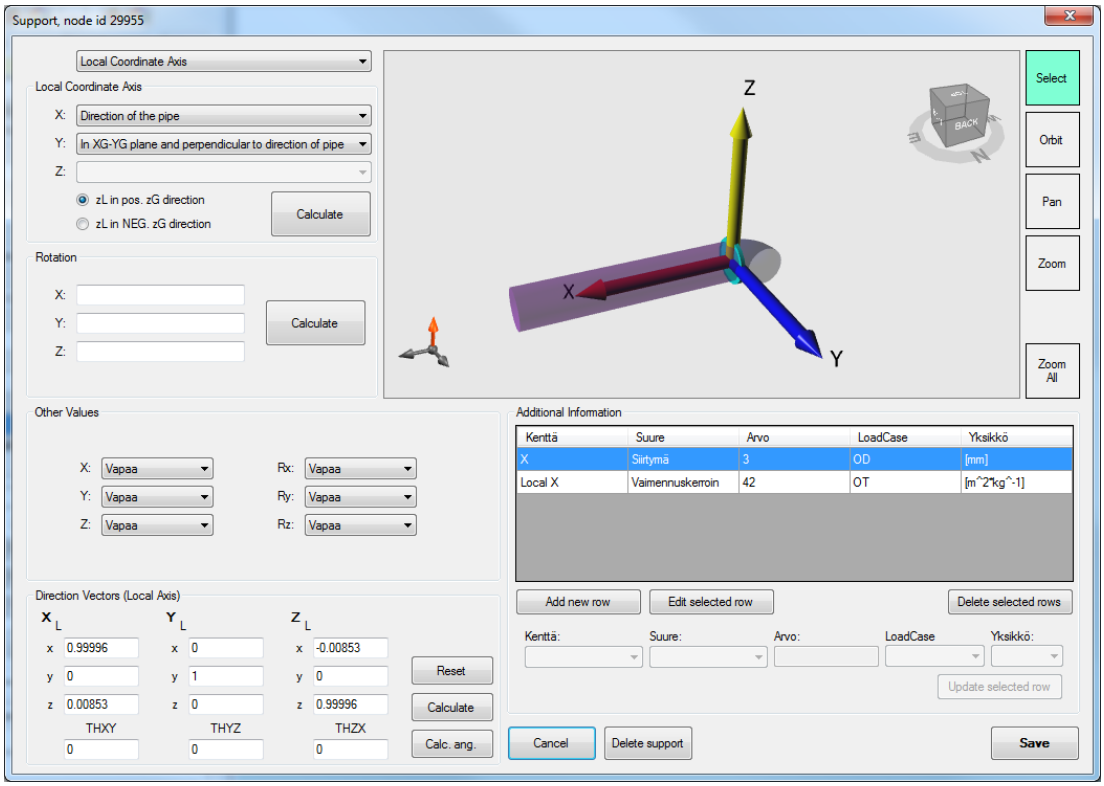
Kaikille alitukityypeille toteutettiin niiden tietokantarakennetta vastaava editointityökalu. Kun käyttäjä painaa jotain näistä napeista, tarkistetaan löytyykö kannasta yhtäkään samansuuntaista ja tyyppistä tukea. Jos yksikin löytyy, käyttäjälle kerrotaan

montako ja kysytään haluaako hän lisätä uuden tuen. Jos yhtäkään ei löydy, voidaan automaattisesti olettaa kyseessä olevan uuden tuen lisäys. Editointityökalun alareunassa sijaitsevien navigointinappien kautta pystytään selaamaan olemassa olevia tukia ja tekemään niihin muutoksia tai poistamaan ne täysin. Tässä työkalussa poikkeuksellisesti uuden tuen tallennus tehdään jo ennen työkalun esittämistä käyttäjälle, jotta tukien välinen navigointi olisi mahdollista. Uudelle tuelle asetetaan tietokantaan oletusarvot ja se ladataan formin tietorakenteeseen samalla tavalla kuin muutkin jo olemassa olevat tuet.

4.2.8 Tuen lokaalivektorien määrittelytyökalu

Tukityökalun esittämiä lokaalivektoreiden muokkaukseen toteutettiin oma työkalunsa, johon pääsee käsiksi kuvassa 18 näkyvän Define local axis-napin kautta. Tuet on tarkoitus piirtää aina putken suuntaisesti ja putket ovat harvoin kohtisuorassa globaalikoordinaatiston kanssa. Siksi tuille on tarpeellista määritellä globaalikoordinaatistosta poikkeava lokaalikoordinaatisto. Tukien lokaalikoordinaatistossa X-suuntavektori on tarkoitus aina osoittaa tuettavan putken suuntaisesti.

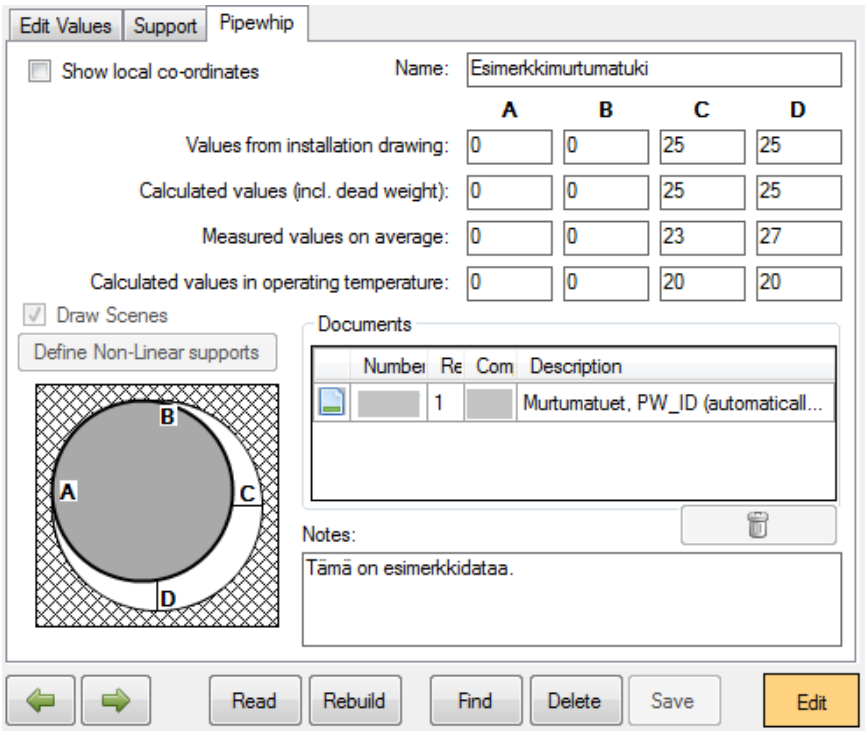
Työkalun latautuessa valittu solmu ja siihen liitetyt elementit piirretään työkalun omaan 3D-kuvaan. Lisäksi kuvaan piirretään nuolisto edustamaan sen hetkistä lokaalikoordinaatistoa, joka tuoreessa tuessa on aina globaalikoordinaatiston suuntainen (Kuva 20). Tätä nuolistoa päivitetään aina, kun koordinaatisto lasketaan uudelleen.



Kuva 20. Lokaalivektoreiden määrittelytyökalu.

4.2.9 Murtumatukityökalu

Murtumatuki on muista tukityypeistä poikkeava tuki. Murtumatuen on tarkoitus tukea putkea vasta murtuman sattuessa. Murtumatuille toteutettiin oma työkalunsa, jonka kautta niitä on mahdollista lisätä, poistaa ja muokata (Kuva 21). Yhdellä solmulla voi olla maksimissaan yksi murtumatuki. Murtumatukiin liittyvä tieto tallennetaan tietokannassa yksinomaan murtumatuille varattuun tauluun.



Kuva 21. Murtumatukityökalun käyttöliittymä.

Työkalun käyttöliittymässä tuelle syötetään välyksien arvot, jotka merkitsevät kuinka kaukana putki on kyseisestä reunasta. Jos esimerkiksi A on 0, on putki silloin kiinni murtumatuen A reunassa. Käyttöliittymässä esitettävän esikatselukuvan päivitys on sidottu välysarvoille tarkoitettujen tekstikenttien ValueChanged-tapahtumaan, eli kuva päivittyy automaattisesti aina kun arvoja muutetaan. Tallentaminen luonnollisesti tapahtuu painamalla Save-nappia, joka on painettavissa ainoastaan, kun muutoksia tietoihin on tehty. Jos valitulla solmulla on murtumatuki, voidaan murtumatuki poistaa painamalla Delete-nappia. Find-nappi on tarkoitettu dokumenttiliitosten luomiseen kyseiseen murtumatukeen ReportBase-ohjelman kautta.

4.2.10 Stress index työkalu

Stress indicet ovat lujuuslaskennassa käytettyjä jännitysintensiteettikertoimia, joilla kerrotaan lujuuslaskentamallilla laskettuja nimellisjännityksiä vastaamaan paremmin todellista jännitystilaa. Putkistoanalyyseissa jännitysintensiteettikertoimia käytetään huomioimaan muun muassa geometrisia (seinämäpaksuuden äkilliset muutokset) tai materiaalin (hitsisaumat) epäjatkuvuuksia, joita ei yleensä voida ottaa huomioon suoraan lujuuslaskentamallissa. Jännitysintensiteettikertoimia määritetään erikseen eri-

tyyppisille kuormituksille, kuten putken sisäinen paine, momenttikuormat ja lämpökuormitukset. (Outinen & Salmi, 2004, 380 – 385; FEMData Oy, 2005)

Stress indicet määritellään solmukohtaisesti putkistotietokantaan ja niiden lisäämiseen, poistamiseen ja muokkaamiseen toteutettiin oma työkalunsa elementtityökalun alle. Työkalun käyttöliittymä on esitettynä kuvassa 22. Käyttöliittymä koostuu tietokantarakennetta vastaavasta tekstilaatikostosta ja taulukosta, johon olemassa olevat stress indicet ladataan. Käyttäjän valitessa olemassa olevan rivin muokkausta varten tai valitessa solmun, johon uutta riviä ollaan lisäämässä, piirretään 3D-kuvaan nuoli osoittamaan valittua solmua.

Kuva 22. Stress index työkalun käyttöliittymä.

Putkistoanalyysia varten käyttäjä määrittää analysoitavan poikkileikkauksen tyypin antamalla siihen liittyvien elementtien komponenttityypin (Itype), päittäishitsin (Gwa) ja pitkittäisen hitsin (Lwa) tunnukset. Jännitysintensiteettikertoimet lasketaan edelleen käyttäjän tekemän tyypityksen ja käytettävän laskentanormin mukaan. Käytettäviä laskentanormeja ovat esimerkiksi ASME NB ja EN-13480.

Rivien yksittäisten lisäysten ja muokkausten sijaan käyttäjä voi halutessaan luoda Excel-tiedoston, johon sisällytetään normikohtaisesti kaikille piirrettyjen elementtien solmujen olemassa oleville stress indiceille oma rivinsä tietoineen. Jos solmulla ei ole yhtäkään olemassa olevaa riviä tietokannassa, lisätään Excel-tiedostoon oletusarvoinen rivi ID:llä "NA” kyseistä solmua varten, johon käyttäjä voi halutessaan täyt-

tää uuden rivin tiedot. Rivien poistaminen onnistuu myös Excel-tiedoston arvojen kautta. Poistettavat tivit merkitään asettamalla ID kenttään *-merkki ID-tiedon jälkeen. Kun käyttäjä on tehnyt haluamansa lisäykset, muokkaukset ja poistomerkinnät, tiedosto hän tallentaa tiedoston haluamaansa paikkaan tuontia varten. Kuvassa 23 on esitettynä pala Excel-tiedoston ohjelmallista käsittelyä C#-kielellä.

```
// Luodaan EXCEL objektit
Microsoft.Office.Interop.Excel.Application xlApp = new Microsoft.Office.Interop.Excel.Application();
if (xlApp == null) { MessageBox.Show("EXCEL could not be started. Check that you have office installed on your system."); return; }
Microsoft.Office.Interop.Excel.Workbook wb = xlApp.Workbooks.Add(Microsoft.Office.Interop.Excel.XlWBATemplate.xlWBATWorksheet);
Microsoft.Office.Interop.Excel.Worksheet ws = (Microsoft.Office.Interop.Excel.Worksheet)wb.Worksheets[1];

// Autofit
xlApp.Cells.HorizontalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;
xlApp.Cells.VerticalAlignment = Microsoft.Office.Interop.Excel.XlHAlign.xlHAlignCenter;
xlApp.Cells.EntireColumn.AutoFit();

// Otsikkojen taustan väritys
range = ws.get_Range("A1:S1");
range.Interior.Pattern = Microsoft.Office.Interop.Excel.XlPattern.xlPatternSolid;
range.Interior.PatternColorIndex = Microsoft.Office.Interop.Excel.XlPattern.xlPatternAutomatic;
range.Interior.ThemeColor = Microsoft.Office.Interop.Excel.XlThemeColor.xlThemeColorDark2;
range.Interior.TintAndShade = -0.249977111117893;
range.Interior.PatternTintAndShade = 0;
```

Kuva 23. Excel-objektien luominen ja otsikkorivin muotoilu ohjelmallisesti.

Tuonti suoritetaan normikohtaisesti, joten käyttäjän tulee valita haluamansa normi alasvetovalikosta ennen tuonnin aloittamista. Tietojen hakeminen Excel-tiedostosta aloitetaan painamalla Import from XLS-nappia. Nappia painettaessa aukeaa Windows-tiedostodialogi, jota käyttämällä käyttäjä hakee haluamansa Excel-tiedoston. Seuraavaksi käyttäjältä kysytään haluaako hän poistaa aiemmat tiedot tietokannasta. Jos käyttäjä vastaa kyllä, poistetaan tietokannasta kaikki valittuna olevaa isometriaa ja normia vastaavat stress index rivit.

Tuontirutiinin aikana kaikki Excel-tiedoston rivit käydään läpi perinteisellä toistorakenteella. ID kentän arvo määrittää miten mikäkin rivi käsitellään. Jos arvo on ”NA”, tietää ohjelma kyseessä olevan uuden rivin lisäys. Jos taas kentästä löytyy *-merkki, on kyseessä rivin poisto ja kaikissa muissa tapauksissa kyseessä on olemassa olevan rivin päivitysoperaatio. Tuonnin valmistuttua käyttäjälle esitetään viesti, joka kertoo operaation valmistuneen ja kehottaa seuraavaksi suorittamaan Calculate All-operaation.

Calculate All-nappia painettaessa haetaan tietokannasta kaikki valitun isometrian elementit ja sitten niihin liittyvät stress index ID:t. Seuraavaksi käydään jokainen ID

läpi yksitellen ja suoritetaan samat arvojen laskentarutiinit, kuin yksittäisiä stress indiceitä lisättäessä, jokaiselle ID:tä vastaavalle stress index riville tietokannassa. Tiedot tallennetaan suoraan kantaan. Tämä operaatio on erityisen hyödyllinen Excel-tuonnin jälkeen, sillä luotavaan Excel-tiedostoon on tarkoitus syöttää ainoastaan osa arvoista, joiden pohjalta loput määrittyvät laskentarutiinien kautta.

5 YHTEENVETO

Opinnäytetyö oli mielestäni haastava, laaja ja mielenkiintoinen. Vaikka olin jo entuudestaan työskennellyt kyseisen projektin parissa, löytyi siitä jatkuvasti uusia haasteita. Pääasiassa itsenäisenä työskentelynä tehdyn työn ja suhteellisen vapaiden käsien myötä sain mahdollisuuden rakentaa omaa osaamistani itseäni kiinnostavaan suuntaan.

Topologyn kokoisen koodikirjaston hallinnan kautta opin paljon laajojen koodikirjastojen hallinnasta. Mitä pidempään tämän projektin kehittymistä olen päässyt seuraamaan, sitä selvemmäksi dokumentoinnin tärkeys ohjelmistoprojekteissa on minulle käynyt. Laajoissa projekteissa, joissa kehittäjät saattavat vaihtua jopa kesken jonkun kokonaisuuden kehittämistä jättäen työn jatkettavaksi seuraavalle, on elintärkeää, että uudelle kehittäjälle jätetään joku ohjenuora johon tarttua. Vaaditaan järjestelmällisyyttä, yhteiset pelisäännöt ja käytännöt.

Niinkin yksinkertainen asia kuin yhden kommenttirivin kirjoittaminen epäselvältä näyttävän koodin viereen selkokielisesti selittäen mitä koodi tekee vie koodin kirjoittajalta joitakin sekunteja, kun taas saman kommentoimattoman koodin selvittely saattaa pahimmassa tapauksessa viedä seuraavalta kehittäjältä tunteja. Joissain tapauksissa koodin koukerot voivat olla niinkin vaikeaselkoisia, ettei seuraavan kehittäjän auta kuin kirjoittaa koko rutiini uudestaan alusta alkaen.

Mielestäni kokonaisuudessaan työ onnistui hyvin, vaikka loppujen lopuksi projekti ei edennyt niin pitkälle, kuin työn aloittaessani kuvittelin sen etenevän. Ei niinkään sik-

si, että eteen olisi tullut ongelmia, vaan enemmänkin koska työtä oli jäljellä suurempi määrä kuin osasin odottaa. Osat, jotka ehdin työn aikana toteuttaa, toimivat luotettavasti ja koodi on rakennettu tulevaa kehitystyötä palvelevalla tavalla. Projektin komponenttien valmiusasteista saatiin myös tilannekartoituksen myötä selvä kuva, joka tulee auttamaan tulevan kehityksen keskittämistä tärkeimmille osa-alueille.

LÄHTEET

DevDept Software, 2016. Viitattu 9.4.2016. https://www.devdept.com/

Ecma International, 2006. C# Language Specification. Viitattu 9.4.2016. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Microsoft, 2016a. Viitattu 10.4.2016. https://www.visualstudio.com/

Microsoft, 2016b. Microsoft API and reference catalog. Viitattu 9.4.2016. https://msdn.microsoft.com/en-us/library/

Microsoft, 2016c. Microsoft tuen elinkaari. Viitattu 10.4.2016. https://support.microsoft.com/fi-fi/lifecycle/search?sort=PN&alpha=Visual%20Basic%206&=Filt

Chauhan S., 2014. Understanding .NET Framework 4.5 Architecture. Viitattu 9.4.2016. http://www.dotnet-tricks.com/Tutorial/netframework/NcaT161013-Understanding-.Net-Framework-4.5-Architecture.html

Bercero C., 2009. What is CLR (Common Language Runtime). Viitattu 9.4.2016. http://carlosbercero.com/post/?post=What_is_CLR_%28Common_Language_Runtime%29

IBM, 2016. IBM Knowledge Center. Viitattu 10.4.2016. http://www.ibm.com/support/knowledgecenter/

W3Schools, 2016. Viitattu 10.4.2016. http://www.w3schools.com/

Williams L, 2006a. White-Box Testing. Viitattu 10.4.2016. http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf

Williams L, 2006b. Testing Overview and Black-Box Testing Techniques. Viitattu 10.4.2016. http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf

TVO 2013. Unique Analysis Capabilities. Viitattu 15.4.2016. http://www.tvo.fi/news/257

Outinen, H. & Salmi, T. 2004. Lujuusopin perusteet. Tampere: Pressus Oy.

FEMData Oy, 2005. Fpipe-ohjelman ASME NB jälkikäsittelijä referenssimanuaali. PDF-dokumentti. Viitattu 28.4.2016.

Ohjelmointiputka, 2007. Ohjelmoijan matematiikka: Osa 3 – Vektorit. Viitattu 28.4.2016. http://www.ohjelmointiputka.net/oppaat/opas.php?tunnus=mat3