jamk.fi

CNC-koneen tiedostonhallinnointi

Wenkura FS

Olli Nissinen

Opinnäytetyö
Huhtikuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Nissinen, Olli	Opinnäytetyö, AMK	24.04.2018
	Sivumäärä 37	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
CNC-koneen tiedostonhallinnointi
Wenkura FS

Tutkinto-ohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Luostarinen Hannu, Raija Hämäläinen

Toimeksiantaja(t)
Yes Neon Oy

Tiivistelmä

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa CNC-koneen tiedoston hallinnointijärjestelmä Yes Neon Oy:lle. Sovellus suunniteltiin vähentämään materiaalihävikkiä, lisäämään työtehokkuutta, vähentämään manuaalisia toimenpiteitä sekä integroimaan eri järjestelmien välisiä työtiedostoja.

Sovellus toteutettiin Windows 7:lle työpöytäsovelluksena. Projektin toteutuskielenä käytettiin C#. Sovelluskehityksessä pyrittiin noudattamaan Microsoftin suosituksia, kuten kolmikerrosarkkitehtuuria ja tehokkaan sovelluksen käyttöliittymän määritelmää.

Toiminnallisuudet toteutettiin käyttäjätarinoiden perusteella. Toteutettavia toiminnallisuuksia olivat mm. aloituspisteiden asettaminen, virhepisteiden poistaminen, referenssiajo, kuvien kääntäminen ja asettelu.

Sovellusta testattiin useammalla testausmenetelmällä. Yksikkötestit toteutettiin MSTestv2 frameworkin avulla. Integraatiotestaus tapahtui Yes Neon Oy:n tiloissa ja hyväksyntätestaus tapahtui toimeksiantajan toimesta.

Sovelluksen käyttöliittymä on suunniteltu helppokäyttöiseksi ja selkeäksi. Käyttöliittymän suunnittelussa on pyritty jakamaan sovellus selkeisiin kokonaisuuksiin. Yleisimmille toiminnallisuuksille on toteutettu pikanäppäinyhdelmät.

Projektin tuloksena toimeksiantaja sai toimivan CNC-koneen tiedoston hallinnointijärjestelmän, joka toteutti käyttäjätarinoiden muodostamat käyttötapaukset. CNC-koneen hallinnointijärjestelmä on osoittautunut toimivaksi ja täyttänyt sille asetetut kriteerit. Asiakas oli tyytyväinen järjestelmän lopputulokseen ja sen toimivuuteen.

Avainsanat (asiasanat)
CNC, ohjelmisto, C#, WPF

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Nissinen, Olli	Bachelor's thesis	24.04.2018
		Language of publication: Finnish
	Number of pages: 37	Permission for web publication: x

Title of publication
File management for CNC machine
Wenkura FS

Degree programme
Software Engineering

Supervisor(s)
Luostarinen Hannu, Hämäläinen Raija

Assigned by
Yes Neon Oy

Abstract

The purpose of the thesis was to design and implement a file management system of CNC machine for Yes Neon Oy. The application was designed to reduce material loss, increase work efficiency, reduce manual actions, and integrate work files between different systems.

The application was implemented for Windows 7 as a desktop application. The project was made with C#. The aim of application development was to use Microsoft recommendations, such as three-layer architecture and powerful application interface definition.

The functionalities were executed based on user stories. The functionalities to be implemented were setting start points, removing exception points, reference drive, rotating and positioning images.

The application was tested by several testing methods. Unit tests were executed with MSTestv2 Framework and integration testing was carried out at Yes Neon Oy. The acceptance testing was performed by the product owner.

The application interface is designed to be easy to use. The aim of the user interface design was to split the application into clear entities. The key features have been implemented with shortcut commands.

As the result of the project, the client received a working CNC file management system that contains use case functionalities. The CNC machine management system has proven to be functional and fulfilled the criteria set for it. The customer was satisfied with the outcome of the system and its functionality.

Keywords/tags (subjects)
CNC, C#, WPF, software

Miscellaneous

Sisältö

Kuviot

Taulukot

Sanasto

C#

C# tarkoittaa C-sharp-ohjelmointikieltä. Kyseinen ohjelmointikieli on vahvasti tyypitetty ja muistuttaa ulkoasultaan muita C-kieliä ja Javaa. C# soveltuu joustavasti moneen eri kehitystyöhön kuten pelikehitykseen, rajapintojen luomiseen, verkkopalveluihin ja pilvipalveluihinkin.

CNC (Computer numerical control)

Yleisesti CNC tarkoittaa koneistettua tapaa muuntaa materiaalista tuote. CNC-koneita ohjataan erillisen sovelluksen kautta, joka lähettää koordinaatteja työstökoneelle.

DLL (Dynamic-link library)

DLL on kirjasto, joka sisältää koodia, jota voidaan käyttää useammassa ohjelmassa samaan aikaan. DLL mahdollistaa koodin uudelleenkäytettävyyden ja modularisoinnin erillisiin komponentteihin.

Git

Git on avoimen lähdekoodin versionhallintajärjestelmä, jota käytetään enimmäkseen ohjelmistokehityksen lähdekoodin hallintaan.

LINQ (Language-Integrated Query)

LINQ on ohjelmointikieleen sisäänrakennettu datan kyselyominaisuus. LINQ:n suurimpia etuja ovat tyypitys sekä johdonmukainen kyselymalli datan käsittelemiseksi eri formaatteihin ja tietolähteisiin.

MDI (Multi-document interface)

MDI mahdollistaa usean dokumentin hallinnoinnin samanaikaisesti sovelluksen käyttöliittymässä. Jokainen dokumentti on erillinen kokonaisuus ja sisältää omat kontrollit. Erilliset kokonaisuudet voivat kommunikoida keskenään pääohjelman kautta.

.NET Framework

.NET Framework on Microsoftin tarjoama ohjelmistokehys joka toimii apuvälineenä sovelluksia rakennettaessa**.**

PLT

Dataformaatti kuvatiedostoille. Voidaan käyttää sekä 2D- että 3D- kuvatiedostojen tallentamiseen. Kuvatiedostot muodostuvat koordinaateista, jotka yhdistetään vektoreilla. Koordinaattien lisäksi PLT-tiedosto sisältää toimintokoodeja piirtämistä varten.

WPF (Windows Presentation Foundation)

WPF on Windows sovellusten esitystapa. Tavallisesti WPF-sovelluksen käyttöliittymä toteutetaan XAML:lla. WPF sisältyy .NET Frameworkiin.

XAML (Extensible Application Markup Language)

XAML on deklaratiivinen merkintäkieli, joka pohjautuu XML-merkintäkieleen. XAML:n käyttötarkoitus on yksinkertaistaa käyttöliittymän luomista. XAML:n avulla voidaan erottaa käyttöliittymä suoritusaikaisesta logiikasta.

XML (Extensible Markup Language)

XML on merkintäkieli, joka on suunniteltu datan siirtoon tai säilöntään. XML suunniteltiin merkintäkieleksi, joka ilmaisee omaa sisältöänsä.

1 Työn lähtökohdat

Opinnäytetyössä oli tehtävänä kehittää tiedostonhallinnointisovellus CNC-konetta varten. Sovellus tulee toimimaan CNC-koneen tiedostonhallinnan käyttöliittymänä. Ohjelmistolla pystytään tekemään monipuolista tiedoston hallinnointia. Sovelluksen toiminnallisuudet mahdollistavat mm. työtiedostojen laajan muokkauksen, CNC-koneen ajosimulaattorin sekä tiedostomuodon ja rakenteen validoinnit. Simulaattorin avulla on mahdollista suorittaa referenssiajo, joka seuraa työstökoneen ajojärjestystä, terän nostokohtia, kulmien terävyyttä, tarkistaa piirtotiedoston virhekohdat ja mahdollistaa niiden korjaamisen. Toteutuksessa pyrittiin myös ennakoimaan ja analysoimaan ajo-ohjelmiston integrointia sekä ohjelmointiympäristöön liittyviä päätöksiä. Työn tilaajana toimi Yes Neon Oy.

2 Yes Neon Oy

Yes Neon Oy on lahtelainen valomainosalan yritys. Toimitilat ovat noin 100 neliömetriä. Yrityksen toimiala liittyy vahvasti mainontaan ja valaistuksiin. Yes Neon on perustettu 1989 ja toiminut nykyisellä yhtiömuodolla vuodesta 1993. Yrityksen tuotteet kohdistuvat pitkälti kauppojen ja yritysketjujen tarpeisiin, kuten teippauksiin, valomainoksiin ja valaistuksiin. Yritys toteuttaa myös ohjelmoitavia ledikylttejä sekä palvelee asiakkaitaan osallistumalla suunnitteluun ja tuotekehitykseen. Yes Neon Oy on tunnettu luotettavana yhteistyökumppanina. (Nissinen 2018.)

3 Sovelluksen käytänteiden esittely

3.1 Suunnittelun käytänteitä

3.1.1 Sovelluksen arkkitehtuurityylejä

Arkkitehtuurinen tyyli parantaa ositusta ja edistää suunnittelun uudelleenkäyttöä tarjoamalla ratkaisuja usein toistuviin ongelmiin. Arkkitehtuurisia tyylejä ja -malleja voidaan ajatella periaatteiksi, jotka muotoilevat sovellusta. Arkkitehtuurityyppien

ymmärtäminen tarjoaa paljon etuja. Tärkein etu on niiden tarjoama yhteinen kieli, mutta ne tarjoavat myös mahdollisuuksia teknologia pohjaisiin keskusteluihin. Tämä helpottaa keskusteluja, jotka sisältävät periaatteita ja malleja ilman erityisiä yksityiskohtia. Taulukossa 1 on yhteenveto tärkeimmistä arkkitehtuurisista tyyleistä. (Chapter 3: Architectural Patterns and Styles 2018.)

Taulukko 1. Avainarkkitehtuuriset tyylit

Arkkitehtuurinen tyyli	**Kuvaus**
Asiakasohjelma / Palvelin	Erottaa sovelluksen asiakasohjelmaan ja palvelimeen. Asiakasohjelma tekee pyynnöt palvelimelle, ja usein palvelin toimii tietokantana, joka sisältää sovelluslogiikan.
Komponenttipohjainen arkkitehtuuri	Hajottaa sovellussuunnittelun toiminnallisiin uudelleenkäytettäviin tai loogisiin komponentteihin, jotka kertovat niille määritellyt viestintärajapintansa.
Toimialapohjainen suunnittelu	Olio-ohjelmoinnin arkkitehtuurinen tyyli keskittittyy liiketoiminta-alueen mallintamiseen ja liiketoimintaobjektien määrittelyyn.
Kerrosarkkitehtuuri	Sovelluksen huolenaiheet jakautuvat kasattuihin ryhmiin eli kerroksiin.
Viestintäketju	Arkkitehtuurinen tyyli, jossa määrätään ohjelmistojärjestelmän käyttö, jolla voidaan vastaanottaa ja lähetetään viestejä monella eri viestintäkanavalla, jotta sovellukset voivat olla vuorovaikutuksessa tietämättä toistensa toimintalogiikasta sen enempää.
N-kerrosarkkitehtuuri / 3-kerrosarkkitehtuuri	Segregoi toiminnallisuudet erillisiin segmentteihin, kuten kerrosarkkitehtuuri, mutta jokainen segementti sijaitsee fyysisesti erillisellä tietokoneella.
Olio-ohjelmoinnin arkkitehtuuri	Suunnittelumalli, joka jakaa sovellusten tai järjestelmien vastuunjaon yksilöidyiksi uudelleenkäytettäviksi tai itsenäisiksi kohteiksi.
Palvelukeskeinen arkkitehtuuri	Kyseisen arkkitehtuurin prosessit ja toiminnnot on suunniteltu toimimaan avoimina, joustavina ja itsenäisinä palveluina.

3.1.2 Käyttöliittymäsuunnittelun suosituksia

Tyypillisesti käyttöliittymän suunnittelua lähestytään käyttäjähaastattelujen, käyttötapausten ja käyttäjäkertomusten pohjalta. Edellä mainittujen avulla pystytään käyttöliittymää suunnitellessa jo nostamaan esille sovelluksen tärkeimmät ominaisuudet. Käyttötapauksien avulla pystytään tasapainottamaan avainasemassa

olevia asioita ja luomaan parempi käytettävyys, yksinkertaistamaan käyttöliittymää sekä lisäämään sovelluksen tehokkuutta. Nämä ominaisuudet koostuvat monista eri osa-alueista ja luovat yhdessä käyttäjäkokemuksen loppukäyttäjälle. (How to design a great user experience for desktop applications 2018.)

Käyttöliittymän yksinkertaisuus saavutetaan valitsemalla oikeat ominaisuudet sekä esittämällä ne käyttäjälle mieluisella tavalla. Oikealla tavalla käytettynä yksinkertaistaminen johtaa helppokäyttöisyyteen. Nämä ovat kuitenkin eri asioita, eikä niitä tule sekoittaa keskenään. Helppokäyttöisyys saavutetaan, kun loppukäyttäjä pystyy tekemään tehtäviä omillaan ilman ongelmia tai sekaannusta kohtalaisessa ajassa. Yksinkertaisuus on tehokkain keino saada aikaan helppokäyttöisyys, ja helppokäyttöisyys on rinnastettavissa käytettävyyteen. Monimutkaisia toimintoja ei juuri käytetä niiden haasteellisuuden vuoksi. Tämän takia yksinkertaiset mallit, jotka suorittavat toimintonsa hyvin ovat paljon käytetympiä. (How to design a great user experience for desktop applications 2018.)

Sovellus voidaan luokitella tehokkaaksi, kun sillä on oikea yhdistelmä seuraavia ominaisuuksia:

- **Mahdollistaa** – Sovellus kykenee täyttämään kohdetyhmiensä tarpeet, joiden avulla he voivat toteuttaa tehtäviä, joita he eivät muutoin voineet tehdä.
- **Tuottavuus** – Sovellus mahdollistaa käyttäjälle suorittaa tehtäviä, jotka parantavat tuottavuutta.
- **Monipuolisuus** – Sovelluksen avulla voidaan suorittaa monia eri tehtäviä tehokkaasti.
- **Suoraviivaisuus** – Sovellus tuntuu auttavan käyttäjää saavuttamaan päämääränsä, ilman turhia toimenpiteitä. Pikakuvakkeet, näppämistön pikavalinnat ja makrot lisäävät suoraviivaisuuden tunnetta.
- **Joustavuus** – Sovellus sallii käyttäjän hienosäätää heidän sovellusympäristöään.
- **Integroitavuus** – Sovellus on hyvin integroitu, jolloin siitä on mahdollista jakaa tietoja muille sovelluksille.
- **Kehittyneisyys** – Sovelluksella on innovatiivisia, moderneja ominaisuuksia ja poikkeuksellisia ominaisuuksia, muihin kilpaileviin sovelluksiin nähden. (How to design a great user experience for desktop applications 2018.)

Osa näistä ominaisuuksista ovat liitännäisiä käyttäjän näkökulmaan sekä sovelluksen nykyisiin ominaisuuksiin. Käsitys siitä, mikä on tehokas sovellus, voi muuttua ajan myötä, jolloin tämän hetken kehittynyt toiminto voi olla huomenna tavallinen. (How

to design a great user experience for desktop applications 2018.)

Käytettävyys muodostuu tehokkuuden ja yksinkertaisuuden tasapainosta. Tyypillisesti turhien elementtien poistaminen, automatisointi, toimintojen luokittelu, toimintojen kuvaukset, elementtien loogiset paikat ja värien käyttö parantavat käytettävyyttä huomattavasti. Asioiden yksinkertaistamisessa piilee kuitenkin riskinsä. Jos yksinkertaistamisen seurauksena käyttäjistä tulee turhautuneita tai hämmentyneitä, on sovelluksen toiminnot pilkottu jo liiankin pieniin osiin, jolloin käytettävyydessä on epäonnistuttu. (How to design a great user experience for desktop applications 2018.)

3.2 Testauskäytänteitä

3.2.1 White-box-testaus

White-box-testaus on menetelmä, jolla testataan sovellusta lähdekoodin tasolla. Tämän testausmenetelmän tarkoituksena on varmentaa lähdekoodin toiminnallisuus. Testeillä pyritään minimoimaan virheet kehitysvaiheessa tarkastelemalla koodin pätkiä, jotka vaikuttavat virheherkiltä. Käytännössä White-box-testaukseen kuuluu koodin testaaminen seuraavia asioita varten:

- Ohjelman sisäiset tietoturva-aukot
- Rikkinäiset tai huonosti rakennetut polut koodausprosesseissa
- Odotettu tulos
- Toimintojen, objektien ja tilan testaaminen yksilöllisesti
- Ehdollisten silmukoiden toiminta. (What is WHITE Box Testing? Techniques, Example & Types 2018.)

White-box-testaamisen toteuttaminen vaatii ymmärrystä lähdekoodista ja sen lopputuloksesta. Useimmiten testaaja ensiksi opettelee ymmärtämään sovelluksen toimintaa ja lähdekoodia. Opettelun jälkeen voidaan rakentaa testipatterit sovellusta varten. (What is WHITE Box Testing? Techniques, Example & Types 2018.)

3.2.2 Black-box-testaus

Black-box-testaustavassa tarkastellaan järjestelmän vaatimuksia ja määrittelyjä. Näiden perusteella testaaja valitsee syötettävät arvot. Syötettyjen arvojen

lopputuloksia vertaillaan odotusarvoihin. Vertailun perusteella tehdään johtopäätökset sovelluksen toiminnallisuudesta. (Ks. kuvio 1.) (What is BLACK Box Testing? Techniques, Example & Types 2018.)

Kuvio 1. Black-box-testaus (What is BLACK Box Testing? Techniques, Example & Types 2018.)

3.2.3 Gray-box-testaus

Gray-box-testaus (ks. kuvio 2.) on tekniikka, jolla testataan sovellusta tietäen osittain sovelluksen sisäisestä toiminnasta. Tämä menetelmä on yhdistelmä sekä White-box-testausta että Black-box-testausta. Gray-box-testauksessa testataan sekä sovelluksen käyttöliittymäkerrosta, että myös lähdekoodiakin. (Gray Box Testing: Process, Techniques, Strategy, Challenges 2018.)

Kuvio 2. Gray-box-testaus (Gray Box Testing: Process, Techniques, Strategy, Challenges 2018.)

4 Projektin suunnittelu

4.1 Toimeksianto

Projektin kohteena oli CNC-koneen työtiedostojen integrointi- ja hallinnointijärjestelmä. Toimeksiantona oli luoda asiakkaalle Windows-ympäristöön sovellus, jolla pystytään integroimaan työtiedostot toisesta järjestelmästä CNC-koneen ajo-ohjelmaan. Sovelluksella tulee pystyä myös kattavasti muokkaamaan työtiedostojen ominaisuuksia ja automatisoimaan niiden sisältöä. Nämä ominaisuudet pitää olla testattuja ja toimintavarmoja.

Tarve kyseiselle ohjelmalle syntyi, koska haluttiin säästää materiaalikuluissa sekä parantaa työtehokkuutta. Pyyntönä oli, että sovellus toimisi täysin offline-tilassa. Tarkempi ominaisuuksien määrittely tapahtui seuraavien käyttäjätarinoiden kautta:

- Käyttäjänä haluan pystyä kääntämään työtiedostoja materiaalin optimoimiseksi.
- Käyttäjänä haluan ohjelman tukevan PLT ja TXT muotoisia tiedostoja.
- Käyttäjänä pystyn asettamaan työtiedostojen aloituspisteen ajotarkkuuden parantamiseksi.
- Käyttäjänä tahdon automatisoida luettavat tiedostot fyysiseen kokoonsa.
- Käyttäjänä tahdon pystyä suorttamaan referenssi ajon ennen oikeaa ajoa.
- Käyttäjänä haluan pystyä poistamaan virhepisteet tiedostosta visuaalisesti.
- Käyttäjänä voin skaalata työtiedoston kokoa.
- Käyttäjänä voin asettaa työtiedostoon liitännäisiä muistiinpanoja.
- Käyttäjänä voin pysäyttää referenssiajon ja halutessani jatkaa referenssiajo loppuun.
- Käyttäjänä pystyn säätämään referenssiajon nopeutta.
- Käyttäjänä pystyn tallentamaan muutokset.
- Käyttäjänä haluan pystyä tarkastelemaan pikanäppäin yhdistelmiä.
- Käyttäjänä haluan etsiä virhepisteitä työtiedostosta.
- Käyttäjänä pystyn tarvittaessa palaamaan lähtötilanteeseen.

4.2 Työn tavoitteet

Toimeksiantajan listaamina päätavoitteina olivat materiaalihävikin vähentäminen sekä työtehokkuuden lisääminen. Näiden toteuttamista varten sovellukseen tuli toteuttaa monia eri toiminnallisuuksia, jotka eivät olleet nykyisillä järjestelmillä mahdollisia. Tällaisia olivat mm. muistiinpanojen kiinnittäminen työtiedostoon, referenssiajon ajonopeuden säätäminen ja erilaisten PLT-tiedostojen integrointi. Uusien ominaisuuksien lisäksi järjestelmään toteutettiin joitakin toimintoja

paranneltuina ja helppokäyttöisempinä. Tällaisia olivat aloituspisteiden merkitseminen ja kuvan asettelun automatisointi. Aikaisemmin kuvan asettelu tapahtui manuaalisesti.

Sovelluksella pyrittiin saavuttamaan muitakin hyötyjä, kuten esimerkiksi kuvien asettelun takia pienenevä materiaalihävikki ja ajojäljen tarkentuminen. Työstettävät materiaalit pyrittiin hyödyntämään paremmin kuvien asettelulla, jolloin myös jätemaksut pienevät. Ajojäljen tarkentamiseksi tavoitteena oli toteuttaa aloituspisteiden asettaminen kuvioon sekä virhepisteiden poistaminen. Virhepisteiden löytämistä varten toteutettiin zoomaus-toiminto.

Nykyisillä sovelluksilla muistiinpanojen kiinnittäminen työtiedostoon ei ollut mahdollista. Kuitenkin muistiinpanot olivat tärkeä osa-alue toteutusta, sillä ilman muistiinpanoja oli työntekijän muistettava CNC-koneen leikkaustapa ja terämalli ulkoa. Muistiinpanojen avulla vähennettiin inhimillisten virheiden mahdollisuutta.

4.3 Käytetyt teknologiat ja ohjelmistot

4.3.1 Ohjelmointikielen valinta

Vertailun kohteeksi valikoituivat kielet jatkokehitys huomioiden. Tärkeimpänä ominaisuutena oli mahdollistaa PCI-7334-kortin ohjaus Texas Instrumentsin tarjoamalla kirjastolla. NiMotion DLL tukee suoraan Visual Basic ja C-kieliä. C# valikoitui mukaan WPF MDI DLL ominaisuuksien vuoksi.

Visual Basic on rivipohjainen ohjelmointikieli. Suurimpia hyötyjä Visual Basicissa on sen kääntäjän matkassa tuleva työkaluvalikoima. Kuten nimestä voidaan päätellä, Visual Basic sisältää paljon visuaalisia elementtejä, ja monet toiminnot ovatkin visuaalisia, kuten käyttöliittymän luominen. Kuitenkin Visual Basicin tukeminen lopetettiin vuonna 2008. Kyseisessä kielessä tyypitys ei ole pakollinen, mutta sen puute saattaa aiheuttaa vaikeasti havaittavia virheitä. Nämä kuitenkin voidaan estää ohjelmamoduulin määrittelyllä. (Blome, Latham, Pratt & Wenzel 2015.)

Positiivisia ominaisuuksia Visual Basicissa on hyvä työkaluvalikoima, NiMotion DLL:n suora tuki jatkokehitystä varten versiossa 6.0 sekä nopea käyttöliittymän luominen. Negatiivisia puolia on heikompi suorituskyky suhteessa muihin vertailussa oleviin ohjelmointikieliin, Visual Basic 6.0:aa ei enää tueta ja Visual Basic .NET ei tue NiMotion kirjastoa. (Blome, Latham, Pratt & Wenzel 2015.)

C-ohjelmointikieli on yleiskäyttöinen, joka on oiva valinta laiteohjelmistojen ja siirrettävien ohjelmien kehittämiseen. C:ssä ei ole juurikaan ohjelmointiin varattuja sanoja, vaan toiminnot tehdään pääsääntöisesti ohjelmointikirjastojen avulla. C:n ydinperiaatteena toimii rakenteellinen proseduraalinen ohjelmointi. Kursailemattomuutensa takia C:stä puuttuu tuki monista muista ohjelmointikielistä löytyville ominaisuuksille, kuten säikeiden käyttö, listojenkäsittely, operaattoreiden ja funktioiden ylikuormitus. Positiivisia puolia C:ssä ovat NiMotion DLL:n suora tuki jatkokehitystä varten, kieli on ollut pitkään käytössä sekä osoittimien käyttö. Negatiivista ohjelmointikielessä on hidas käyttöliittymän luominen ja runsas kirjastojen käyttäminen. (Allain 2018.)

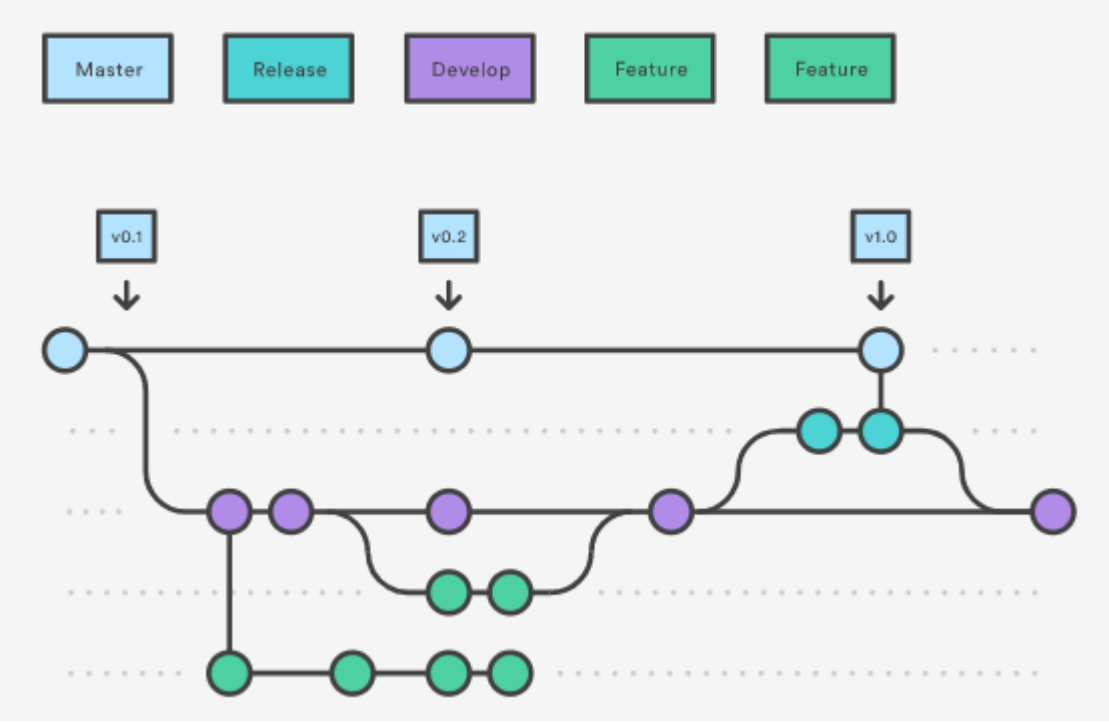
C# on tyyppivarma olio-ohjelmointikieli, jonka avulla kehittäjät voivat luoda turvallisia sovelluksia. Nämä sovellukset käyttävät .NET Frameworkia. C#:n avulla voidaan luoda mm. Windows sovelluksia, verkkopalveluita, pelejä jne. C# .NET Visual Studion kanssa yhdessä tarjoaa edistyksellisen koodinmuokkauksen, käyttöliittymäsuunnittelun työvälineet, integroidun debuggerin ja monia muita työvälineitä, joiden avulla on ketterä kehittää sovelluksia. (Blome, Latham, Levin, Wagner & Wenzel 2015.)

Microsoft tarjoaa erinomaisen dokumentaation ja koodikäytännöt C#:sta. Lisäksi valmiita kirjastoja löytyy erittäin hyvin. Kyseiseen ohjelmointikieleen integroitu tietokantakysely (LINQ) mahdollistaa nopeamman kehityksen ja intelisenssin käytön tietokantakyselyitä tehdessä. Positiivista C#:ssa on tyyppivarmuus, suurin kehittäjäyhteisö vertailluista kielistä, .NET Frameworkin tarjoamat ominaisuudet, nopea käyttöliittymän luonti, WPF MDI DLL, ja kyseinen kieli on myös korkean tason ohjelmointikieli. Negatiivista on, että NiMotion DLL:n käyttö vaatii erillisen wrapperin. (Blome, Latham, Levin, Wagner & Wenzel 2015.)

Vertailussa C jäi nopeasti ulkopuolelle, sillä Visual Basic ja C# tarjosivat enemmän valmiita ominaisuuksia, jotka sopivat hyvin tähän projektiin. Visual Basic ja C# tarjosivat molemmat samankaltaisia etuja. Kuitenkin erottavaksi tekijäksi muodostui tiedon löytäminen, yhteisön määrä sekä oma mieltymys kyseiseen koodikieleen. Näiden tekijöiden perusteella ohjelmointikieleksi valittiin C#.

4.3.2 Git

Lähdekoodin hallinta tapahtuu Gitin avulla. Tämän järjestelmän avulla voidaan tehdä omia versiohaaroja, jotka voidaan yhdistää päähaaraan hallitusti (Branching and Merging 2018). Tässä projektissa käytettiin haaroja uusien ominaisuuksien luomiseen, jotka yhdistetään toiminnallisuuden valmistuessa.



Kuvio 3. Gitin työnkulku esimerkki (Git Workflow 2018)

4.3.3 WPF

WPF:n ydin on resoluutiosta riippumaton vektoripohjainen renderöintimoottori, joka on rakennettu hyödyntämään nykyaikaista grafiikkalaitteistoa. Kyseinen esitystapa mahdollistaa niin 2D- kuin 3D-grafiikoiden käytön. WPF sisältyy .NET Frameworkiin.

Visual Studiossa WPF-sovelluksien kohdalla käyttöliittymä luodaan XAML:lla, joka renderöidään realiajassa tarkasteltavaksi ja muokattavaksi.
(Blome, Jones, Peek, Robertson & Waren 2016.)

4.3.4 Visual Studio 2017

Visual Studio on interaktiivinen kehitysympäristö, jonka avulla voi rakentaa sujuvasti sovelluksia. Se soveltuu niin mobiili-, työpöytä- kuin web-sovellusten luomiseen kaikille käyttöliittymille. Visual Studio sisältää paljon valmiita pohjia, joiden päälle on helppo lähteä rakentamaan sovellusta. Lisäksi sovellukseen on mahdollista asentaa GIT integraatio sekä muita työskentelyä helpottavia lisäosia. Kyseisessä kehitysympäristössä on myös tuki Azuren pilvipalveluissa toimivien sovellusten rakentamiseen. Visual Studio 2017 sisältää .NET Frameworkin, koodianalysaattorin, MSTestv2 Frameworkin sekä monia muita helposti integroitavia ominaisuuksia. (Brown, Jones, Lee, Robertson & Warren 2017.)

4.3.5 MSTestv2 Framework

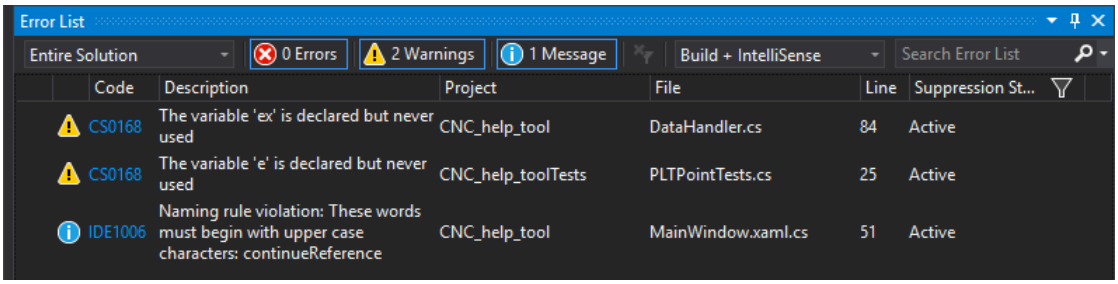
MSTestv2 on yksikkötestaukseen suunniteltu framework. Testitapaukset voidaan ajaa erikseen tai kiinnittää ohjelman kääntövaiheeseen. Kyseisellä frameworkilla voidaan luoda valmiista luokkatiedostoista testitapauksia. MSTestv2 generoi luokille omat testitiedostot. Tiedosto sisältää oman yksikkötestipohjan. (Ks. kuvio 4.)

```
10  namespace CNC_help_tool.Classes.Tests
11  {
12      [TestClass()]
13      public class PLTPointTests
14      {
15          [TestMethod()]
16          public void PLTPointTest()
17          {
18
19          }
20      }
21  }
```

Kuvio 4. MSTestv2:lla generoitu yksikkötesti pohja

4.3.6 Code analysis

Koodianalysaattori (Code analysis) on Visual Studion 2017 ominaisuus tarkkailla koodin laatua. Koodianalysaattoriin on mahdollista määritellä säännöt tai käyttää valmista konfiguraatiota. Näiden asetusten perusteella Code analysis reagoi ja ilmoittaa käyttäjälle virheistä. Oikein määriteltynä analysaattori voi parantaa koodin käytettävyyttä, luettavuutta, turvallisuutta sekä suorituskykyä. Analysaattori voidaan asettaa automaattisesti tai manuaalisesti suoritettavaksi. (Ks. kuvio 5.) (Code Analysis for Managed Code Overview 2018.)



Kuvio 5. Esimerkki koodianalysaattorin huomioista

4.3.7 .NET Framework

.NET Framework on Microsoftin tarjoama ohjelmistokehys, joka toimii apuvälineenä sovelluksia rakennetaessa. Frameworkien tarkoitus on nopeuttaa sovellusten valmistusta. .NET Framework koostuu kahdesta osasta: .NET-kirjastoista sekä sovelluksen ajoympäristöstä, joka kääntää ohjelmakoodin käyttöjärjestelmälle sopivaan muotoon. (Jones, Latham, Petrusha, Pratt & Wenzel 2017.)

4.4 Käyttöliittymän suunnittelu

Koska sovelluksen kohdealustana oli Windows, sovellus päätettiin jakaa useampaan näkymään WPF MDI DLL:n avulla. Koska sovelluksen tuli toimia offline-tilassa, tämä rajasi paljon ominaisuuksia pois käyttöliittymän suunnittelusta. Käyttäjätarinoiden kautta saatiin selkeä kuva ohjelman kokonaisuudesta ja laajuudesta. Sovelluksen monipuolisten toiminnallisuuksien takia päätettiin lähestyä suunnittelua käyttöliittymä edellä. Tällä pyrittiin välttämään sekavan oloinen rakenne, jonne on

lisätty ominaisuuksia jälkikäteen. Käyttäjätarinat jaoitteltiin ryhmiin, joista jokainen ryhmä muodostaisi oman näkymänsä. Näkymät olisivat muokattavissa ja piilotettavissa käyttäjän mielen mukaan.

Sovelluksen ydin on selkeästi käyttäjätarinoiden jaottelun perusteella piirto- ja hallinnointinäkymät. Vaikka näkymät eivät olekaan tietoisia toisistaan, ne ovat silti linkityksissä toisiinsa päänäkymän kautta. Tällöin päänäkymä voi hoitaa tarvittaessa linkittämisen. Näkymät muodostuivat käyttäjätarinoista seuraavalla jakaumalla:

Päänäkymä

- Käyttäjänä haluan ohjelman tukevan PLT- ja TXT-muotoisia tiedostoja.
- Käyttäjänä tahdon automatisoida luettavat tiedostot fyysiseen kokoonsa.
- Käyttäjänä pystyn tallentamaan muutokset.

Piirtonäkymä

- Käyttäjänä haluan etsiä virhepisteitä työtiedostoista visuaalisesti.
- Käyttäjänä pystyn tarvittaessa palaamaan lähtötilanteeseen.

Hallinnointinäkymä

- Käyttäjänä haluan pystyä kääntämään työtiedostoja materiaalin optimoimiseksi.
- Käyttäjänä pystyn asettamaan työtiedostojen aloituspisteen ajotarkkuuden parantamiseksi.
- Käyttäjänä haluan pystyä poistamaan virhepisteet tiedostosta.
- Käyttäjänä voin skaalata työtiedoston kokoa.
- Käyttäjänä voin pysäyttää referenssiajon ja halutessani jatkaa referenssiajo loppuun.
- Käyttäjänä tahdon pystyä suorttamaan referenssiajon ennen oikeaa ajoa.
- Käyttäjänä pystyn säätämään referenssiajon nopeutta.

Ohjenäkymä

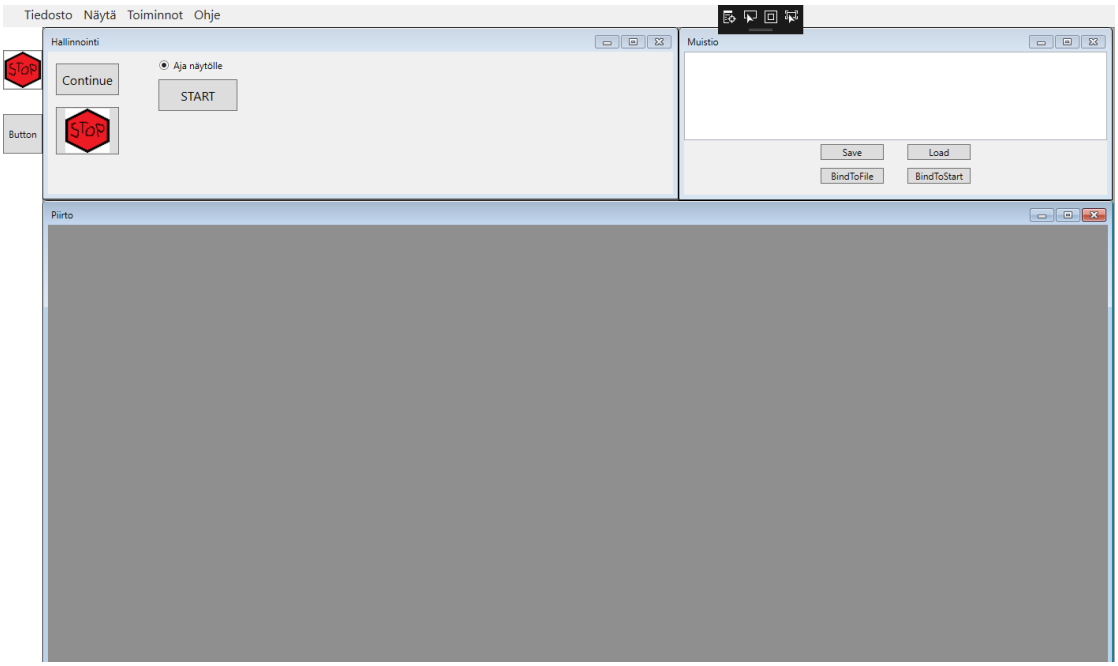
- Käyttäjänä haluan pystyä tarkastelemaan pikanäppäinyhdistelmiä.

Muistionäkymä

- Käyttäjänä voin asettaa työtiedostoon liitännäisiä muistiinpanoja.

WPF MDI DLL:n avulla päätettiin muodostaa näkymät päänäkymän sisään. Tällöin päätason kontrollit ovat aina käytettävissä ja muut näkymät muokattavissa sen sisällä. Erillisten näkymien muodostaminen mahdollistaa myös helposti uuden kokonaisuuden lisäämisen sovellukseen. Jatkokehityksen kannalta on tärkeää, että näkymien ja niiden sisällä olevien ominaisuuksien ei tarvitse olla tietoisia toisistaan.

Kuviossa 6 on toteutettu hahmotelma, miltä sovelluksen käyttöliittymä voisi näyttää. Hahmotelmassa on jaoteltu näkymät omiin MDI-komponentteihinsa.

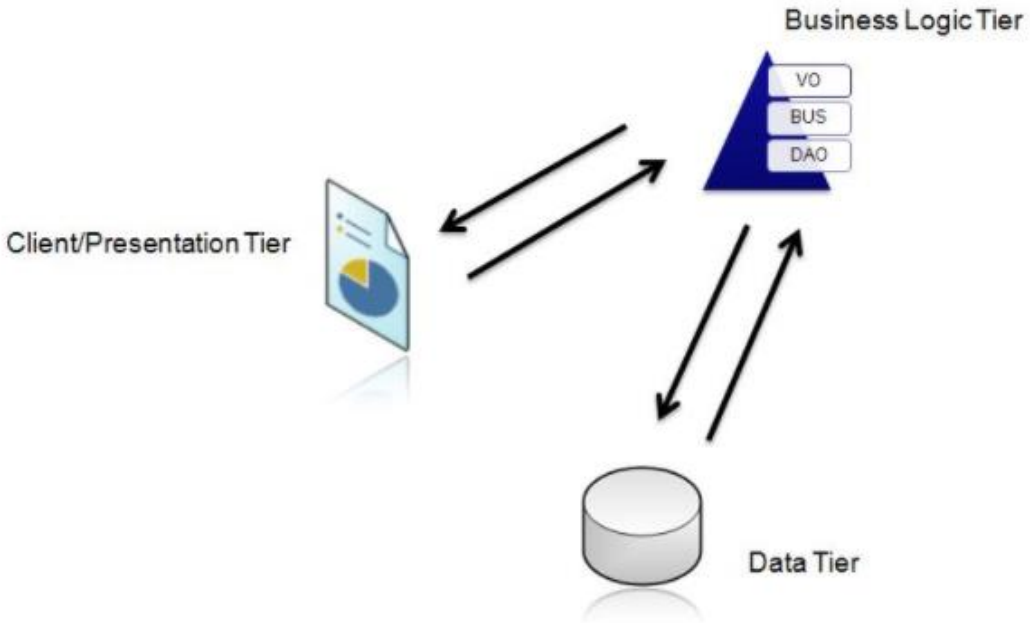


Kuvio 6. Hahmotelma käyttöliittymästä

4.5 Ohjelmiston arkkitehtuurin suunnittelu

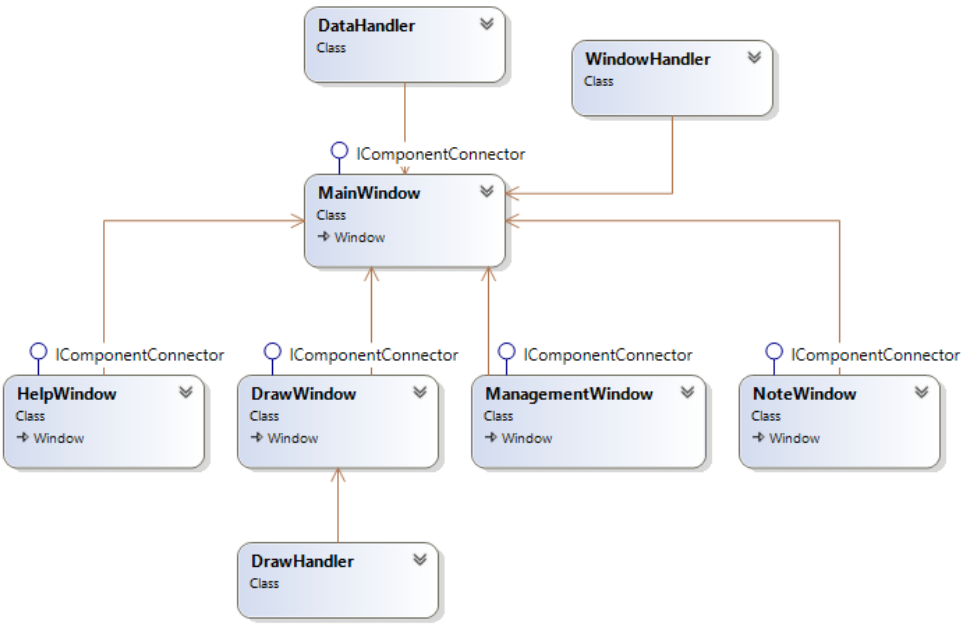
Offline-tilansa takia arkkitehtuurisista ratkaisuista voitiin rajata suuri osa pois. Jatkokehitys huomioiden arkkitehtuurisia ratkaisuja mietittiin siten, että sovellus voitaisiin siirtää helposti online-tilaan. Koska sovellus on vahvasti kytköksissä toimialaan, päätettiin sovellus toteuttaa olio-ohjelmointipohjaisen toimiala-arkkitehtuurin mukaisesti, johon on yhdistty kerrosarkkitehtuuri erottelemaan data-,

business- ja esityskerrokset. (Ks. kuvio 7.)



Kuvio 7. Kerrosarkkitehtuuri (Parikshit 2014)

Käyttäjätarinoiden pohjalta pystytään päättelemään pitkälti, minkälaisia luokkia ja näkymiä ohjelmassa tulisi olla. Monipuolisten toiminnallisuuksien tulisi olla jaoteltavissa loogisiin lohkoihin käyttöliittymän suunnitteluvaiheessa, jolloin ne on myös huomioitava arkkitehtuurisissa ratkaisuissa. Kuviossa 8 on hahmotelma ohjelman rakenteesta UML-kaaviona.



Kuvio 8. Hahmotelma sovelluksen rakenteesta

4.6 Testauksen suunnittelu

Sovelluksen testaamisessa hyödynnettiin white-box- ja black-box-testauksista tuttuja menetelmiä. Sovellukselle tehtiin luokkakohtaiset testitapaukset. Luokkien metodeille toteutettiin yksikkötesti, jotka automatisoitiin ajettaviksi sovelluksen kääntövaiheessa. Toiminnallisuuden varmentaminen tapahtui kehittäjän toimesta. Varmentamisesta luotiin selkeä kuvaus testatuista ominaisuuksista ja testausajankohdista. Hyväksymistestaus tapahtui toimeksiantajan toimesta. Testaus toteutettiin MSTestv2 Frameworkillä, joka on integroituna Visual Studioon.

5 Toteutus

5.1 Toiminnallisuudet

5.1.1 Käyttöliittymän toteutus

Käyttöliittymän toteutuksessa on pyritty noudattamaan Microsoftin tarjoamia käyttöliittymäsuunnittelun suosituksia. Sovelluksella on pyritty saavuttamaan mahdollisimman monta kohtaa aikaisemmin esitetyn tehokkaan sovelluksen määritelmästä. Erityisesti helppokäyttöisyyteen ja suoraviivaisuuteen on panostettu. Näkymistä on pyritty tekemään mahdollisimman selkeitä ja kuvaavia käyttäjälle. Toiminnallisuuksiin on kytkettynä pikanäppäimiä lisäämään suoraviivaisuutta ja työtehokkuutta.

WPF MDI-käyttöliittymän rakentaminen toimii erillisten komponenttien kautta. Komponentit olivat tässä projektissa ikkunapohjaisia, mikä tarjoaa selkeän erittelyn eri toiminnallisuuksia tarjoavien kategorioiden välille. Nämä komponentit, mahdollistavat näkymien muokkaamisen käyttäjän mielen mukaisesti. Kyseisen sovelluksen käyttöliittymä on toteutettu WPF:n tarjoamalla XAML:lla, joka tarjoaa kehitysvaiheessa reaaliaikaisen näkymän siitä, miltä kyseinen komponentti tulisi näyttämään. Kyseinen ominaisuus mahdollistaa erittäin ketterän tavan ulkoasun muokkaamiseen. XAML:ssa elementeille voidaan suoraan kytkeä toiminnallisuuksia,

datasidoksia sekä antaa määreitä, joita voidaan hyödyntää bisneslogiikan puolella. (Ks. kuvio 9.)

```
39              <StackPanel Orientation="Vertical" Width="150">
40                  <Button
41                      Width="80" Margin="10,5, 0, 0" Height="40" Content="Jatka" FontSize="16"
42                      Name="continueReferenceDrive" Click="ContinueReferenceDrive_Click" >
43                  </Button>
44                  <Button
45                      Width="80" Margin="10,10, 0, 0" Height="40" Content="Pysäytä" FontSize="16"
46                      Name="stopReferenceDrive" Click="StopReferenceDrive_Click" >
47                  </Button>
48              </StackPanel>
```

Kuvio 9. Toiminnallisuuden kiinnittäminen painikkeeseen

WPF MDI DLL mahdollistaa muiden ikkunanäkymien kiinnittämisen pääikkunan sisälle. Normaalisti erilliset ikkunanäkymät ovat siirrettävissä pääikkunan ulkopuolelle, mutta kyseinen kirjasto tarjoaa mahdollisuuden estää tämän. Myös muut ominaisuudet kuten pienennetyt ikkunat ja koon määrittäminen pysyvät päänäkymän sisällä. Näkymien keskeinen data linkitys on toteutettu kulkemaan päänäkymän kautta. Esimerkiksi muokkausominaisuuden määrittelemät komennot ovat linkitettyinä päänäkymän kautta piirtonäkymälle. (Ks. kuvio 10.)

```
1   <UserControl x:Class="CNC_help_tool.SubWindows.DrawWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       Height="600"
5       Width="1000"
6       Background="#FFFFFF"
7       Name="table"
8       Loaded="BindKeyboard_Shortcuts">
9
10      <Grid>
11          <Canvas Background="#FF8F8F8F"  Margin="0,0,0,0" Name="myCanvas" ClipToBounds="True"
12                  MouseLeftButtonDown="MouseLeft_click" MouseRightButtonDown="MouseRight_click" MouseWheel="MouseWheel_scrolled" >
13              <Canvas.LayoutTransform>
14                  <ScaleTransform ScaleY="-1"></ScaleTransform>
15              </Canvas.LayoutTransform>
16          </Canvas>
17      </Grid>
18  </UserControl>
```

Kuvio 10. WPF MDI:n määritteleminen XAML:lla

5.1.2 Kuvion piirtäminen

Kuvan piirtoalustana toimii WPF:n Canvas, joka mahdollistaa paljon ominaisuuksia piirtämistä ja koordinointia varten. Kuvan piirtäminen on toteutettu aliobjektien pohjalta. Näiden toiminta tapahtuu indeksoidussa järjestyksessä.

Itse kuvan muodostaminen on toteutettu piirtämällä viiva jokaisen aliobjektin sisältämän pisteen välille, jolloin aliobjekti alkaa ja loppuu samaan pisteeseen (ks. liite 1). Tällöin jokainen aliobjekti muodostaa oman kuvionsa, joista kuva kokonaisuudessaan muodostuu. Kuvan piirtäminen on toteutettu tiedoston valinnan yhteydessä validoinnin onnistuessa. Virhetilanteissa käyttäjälle ilmoitetaan PLT-formaattivirheistä.

5.1.3 Referenssiajo

Referenssiajon avulla on mahdollista tarkastella kyseisen työtiedoston toiminnallisuutta. Ajosimulaattori simuloi, kuinka CNC-kone tiedoston ajaisi. Tarkkailun kohteena ovat pääsääntöisesti ajojärjestys ja mahdolliset virhepisteet. Referenssiajo käytännössä piirtää PLT-tiedoston kuvan piste kerrallaan.

Referenssiajo on mahdollista pysäyttää missä vaiheessa tahansa. Ajon pysäytyksen aikana on mahdollista tarkastella osittain piirrettyä kuvaa, jatkaa referenssiajoa tai aloittaa se alusta. Simulaattorin ajonopeuden muuttaminen onnistuu niin uutta ajoa aloittaessa tai ajon aikana. Myös nämä toiminnot ovat validoituja.

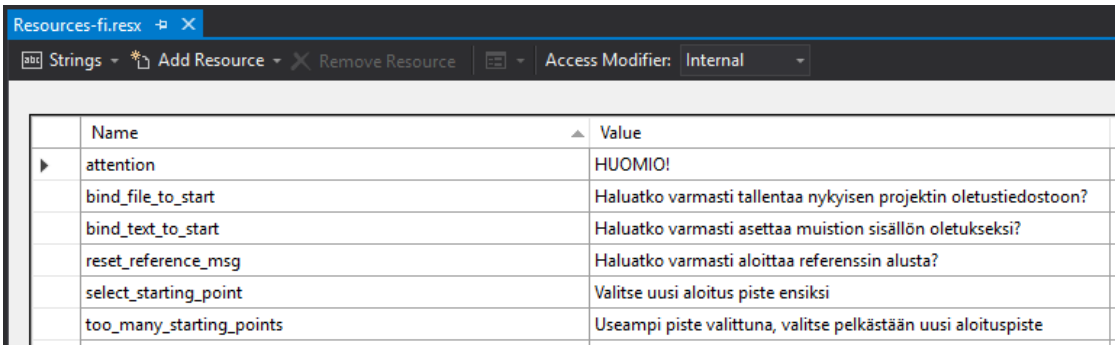
Toteutukseltaan referenssiajo on hyvinkin lähellä kuvion piirtämistä. Käytännössä kuvion piirtäminen on itse tietoinen siitä, piirretäänkö referenssiä vai ei. Referenssiajo vain säätelee kuvion piirtämistä erilaisilla muuttujilla ja täten vaikuttaa sen toimintaan. Ainoana erona kuvion piirtämisessä on piirron tapahtuminen säikeissä. (Ks. kuvio 11.)

```
477     public void CallDriveReference(Canvas myCanvas, int driveSpeed)
478     {
479         if (drivingReference == true && paused == true)
480         {
481             // asks user to confirm that he wants to start from beginning
482             if (MessageBox.Show(rm.GetString("reset_reference_msg"), rm.GetString("attention"),
483                 MessageBoxButton.YesNo, MessageBoxImage.Question) == MessageBoxResult.Yes)
484             {
485                 drivingReference = false; paused = false;
486             }
487         }
488         if (drivingReference == false)
489         {
490             DrawPicture(myCanvas, driveSpeed, true);
491         }
492     }
```

Kuvio 11. Referenssiajo

5.1.4 Lokalisointi

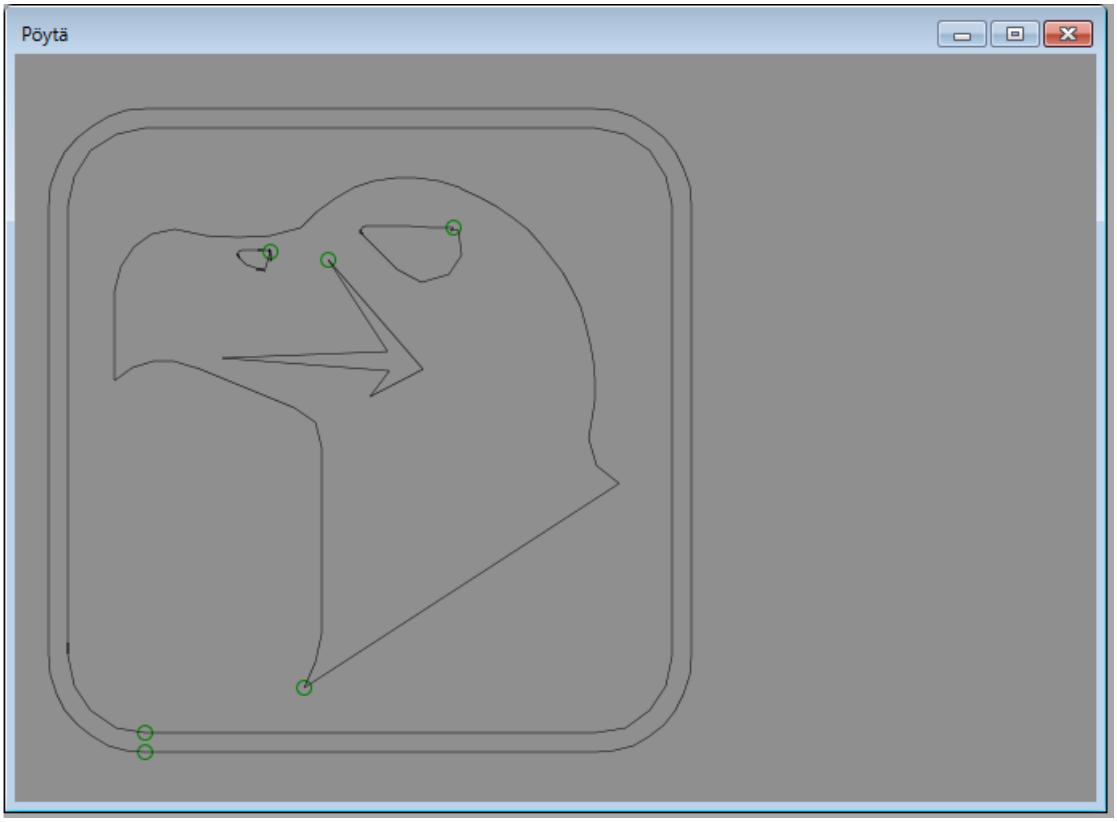
Sovelluksen ilmoitukset ja tekstit ovat toteutettu pääsääntöisesti resurssitiedostojen avulla. Tämä mahdollistaa tarvittaessa sovelluksen kääntämisen helposti toiselle kielelle. Resurssit ovat määritelty avaimien perustella, jolloin resurssin valinta tapahtuu kulttuuritiedon kautta. Resurssit ovat kiinnitettyinä järjestelmän globaaliin resurssimanageriin ja ovat käytettävissä sen kautta. (Ks. kuvio 12.)



Kuvio 12. Resurssitiedoston määritteleminen

5.1.5 Kuvion muokkaaminen

Kuvan muokkaamiseen toteutettiin useita eri ominaisuuksia. Ominaisuuksien tarkoituksena oli lisätä työtehokkuutta ja vähentää materiaali hävikkiä. Jokaisella kuvan aliobjektilla on oma aloituspiste. Aloituspiste on merkattuna vihreällä. Tällöin kuvan ajojärjestyksen asettelu on mahdollista ja sen avulla pystytään välttämään kappaleiden irtoaminen levystä ennen aikojaan. Oikein asetetulla aloituspisteellä vältetään kappaleiden heilumisesta aiheutuva epätarkkuus. (Ks. kuvio 13.)

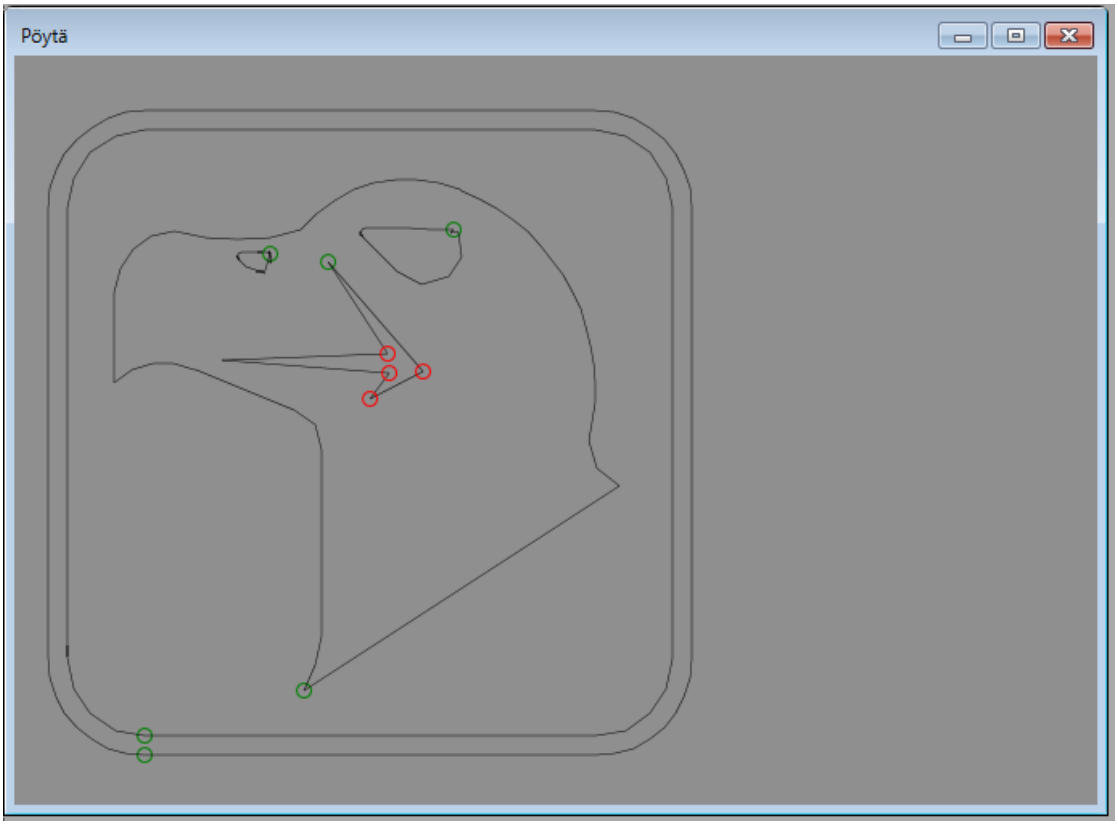


Kuvio 13. Aloituspisteiden merkkaaminen

Kuvasta voidaan poistaa tarpeettomat tai virheelliset pisteet, kuten kulmissa muodostuvat kolmiot, jotka aiheuttavat epätarkkuutta ajojälkeen. Pisteiden poistaminen voidaan toteuttaa yksittäin tai poistamalla useampi piste kerralla. Poistamista varten valitut pisteet ovat merkattuina punaisella. Myös kokonaisten aliobjektien poistaminen on mahdollista.

Pyöreiden muotojen pisteiden karsiminen nopeuttaa CNC-koneen ajoaikaa huomattavasti. Pisteiden poistamista varten sovellus sisältää myös palautusominaisuuden. Tämä mahdollistaa käyttäjän palaamisen takaisin lähtötilanteeseen virheellisen toiminnon sattuessa. (Ks. kuviot 14-15.)

Kuvio 14. Zoomilla löytynyt virhepiste-esimerkki



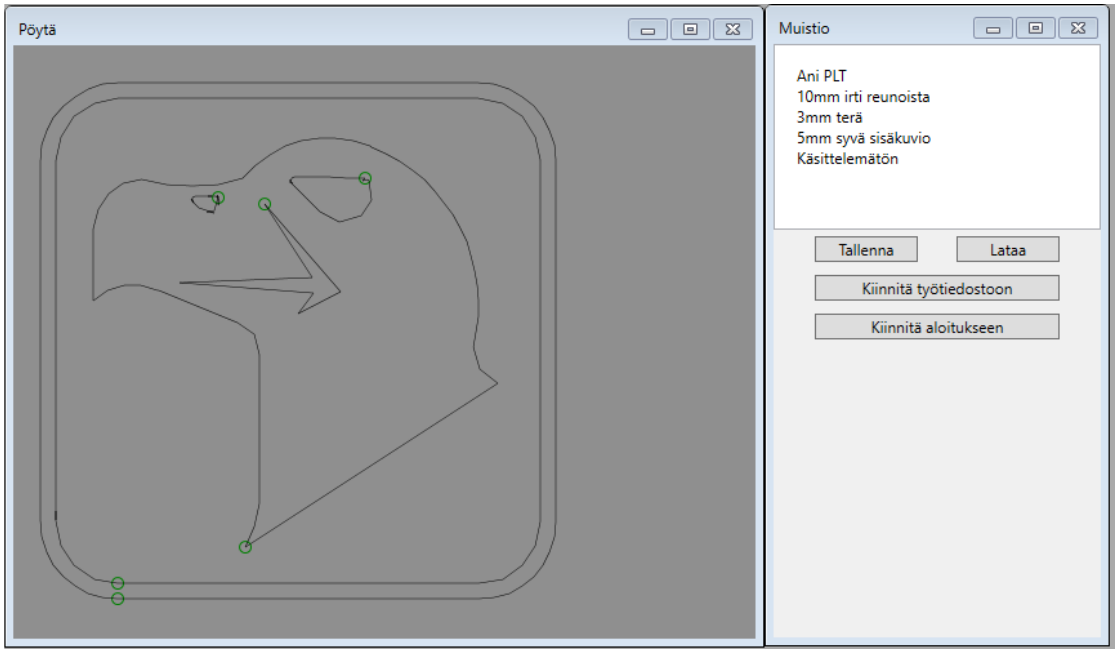
Kuvio 15. Pisteiden valitseminen ja poistaminen

Kuvaa voidaan käännellä halutun asteluvun mukaisesti. Kääntämistä varten kuva siirretään origoon, jonka jälkeen kuva siirretään takaisin lähtöpisteeseen. Jos kääntämisen jälkeen kuva on osittain ulkona työstöpöydältä, asetetaan se takaisin työstöalueelle käyttäen aikaisempia marginaaleja. Kuvien kääntely ja asettelu mahdollistavat materiaalin optimoinnin tehostamisen. Muokkaus sisältää prosentuaalisen muokkausmahdollisuuden. Tämän avulla on tarvittaessa mahdollista asettaa X- ja Y-akselit eri mittasuhteisiin. Kaikkia muokkausominaisuuksia voidaan käyttää samaan aikaan, ilman ongelmia. Muokkausten epäonnistuessa on mahdollista palata takaisin aloitustilanteeseen.

5.1.6 Automatisointi

Sovelluksen tarkistettua luetun tiedoston data, työstettävä kuva pyritään automatisoimaan mahdollisuuksien mukaan. Ladatun tiedoston ollessa mittasuhteiltaan suurempi kuin CNC-koneen fyysinen kapasiteetti hoidetaan kuvan koon asettelu pöytään nähden sopivaksi. Kyseinen ominaisuus on tarvittaessa konfiguroitavissa. Koon lisäksi kuvan sijainti pyritään asettelemaan origon läheisyyteen materiaalin säästämiseksi. Reunoihin jätetään pieni työstövara, ettei leikattava kuva irtoa liian aikaisin ja pääse heilumaan CNC-koneen työstöpöydällä. Tämän takia FlexiSIGN -ohjelmassa ei tarvitse tarkastella kuvion absoluuttista sijaintia, koska kuvan sijainti automatisoidaan.

Työtiedostoihin on mahdollista kiinnittää muistiinpanot. Muistiinpanojen liittäminen työtiedostoon tapahtuu nappia painamalla. Sovellus lukee kyseiset työtiedostoon kiinnitetyt muistiinpanot automaattisesti ja avaa ne sivulle katsottaviksi. Sovellukseen on mahdollista kiinnittää oletusmuistiinpanot, jotka avataan aina sovelluksen yhteydessä. Nämä ominaisuudet ovat konfiguroitavissa käyttäjän niin halutessa. (Ks. kuvio 16.)



Kuvio 16. Työtiedostoon kiinnitetyt muistiinpanot

5.1.7 Kuvion tarkistaminen

Piirrettyä kuvaa on mahdollista tarkastella erilaisilla ominaisuuksilla. Zoomauksen avulla voidaan tutkia mahdollisia virhepisteitä tai tallentaa zoomattu, relatiivinen koko. Kuviosta on mahdollista tarkistella kaikkia pisteitä yhtäaikaisesti. Tällöin päällekkäin merkatut tuplapisteet voidaan poistaa. Pisteiden tarkastelun yhteydessä voidaan tarkistaa aloituspisteiden sijainnit jokaisesta aliobjektista. Kuvion validointi tapahtuu automaattisesti taustalla ja järjestelmä ilmoittaa virheistä käyttäjälle. Validointi on toteutettu useassa eri portaassa. Datan muoto ja formaatti validoidaan tiedostoa luettaessa. Kun datasta muodostetaan aliobjektit, validoidaan jokaisen pisteen sisältö luontivaiheessa. Kuvion muokkauksen yhteydessä toiminnot ja syötteet varmistetaan formaatin säilymisen ja toiminnallisuuden vuoksi. Tallennuksen yhteydessä vielä tarkistellaan data ennen tiedostoon kirjoitusta.

5.2 Testaus

Testaus on toteutettu white-, gray- ja black-box-menetelmillä. White- ja gray-box-testaus tapahtui tekijän toimesta ja black-box-testaus tuoteomistajan toimesta. White-box-testaus toteutettiin MSTestv2 Frameworkin yksikkötesteillä. Funktioille on luotu yksikkötestit, joiden avulla pystytään varmentamaan funktioiden toiminnallisuus. Yksikkötestit noudattavat käytänteiden mukaista “pystytä ja tuhoa” -menetelmää. Tässä menetelmässä jokaiselle yksikkötestille pystytetään tarvittava rakenne testin ajamiseen. Tällä tavalla pystytään tarjoamaan oikean kaltainen testiympäristö. (Ks. kuvio 17.)

```
12      [TestClass()]
13      public class DataHandlerTests
14      {
15          [TestMethod()]
16          public void ReadPLTPointFromDataTest()
17          {
18              DataHandler dataHandler = new DataHandler();
19              List<PLTPoint> coordinatePoints = dataHandler.ReadPLTPointFromData("../../TestMaterials/ANI001.plt");
20              if (coordinatePoints.Count != 193)
21              {
22                  Assert.Fail();
23              }
24              Assert.IsNotNull(coordinatePoints);
25          }
26      }
```

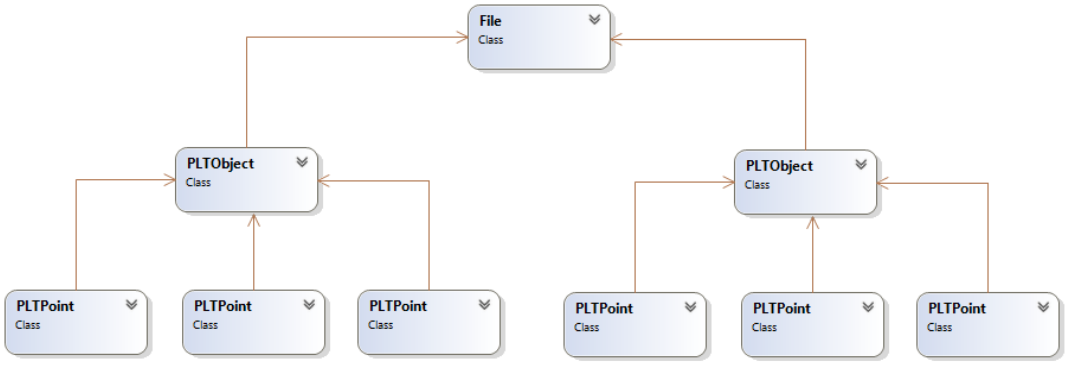
Kuvio 17. Esimerkki yksikkötestistä

Gray-box-testauksella on pyritty varmentamaan käyttöliittymän ja funktioiden toiminallisuutta. Myös sovellusten väliset integraatiot ovat testattuja ominaisuuksia. Näiden testaus on tapahtunut gray-box-menetelmiä hyödyntäen. Toiminnallisuuksien varmentamisen jälkeen tuoteomistaja on hoitanut hyväksyntätestauksen, joka on osa black-box-testausta. Hyväksyntätestauksessa on tarkasteltu syötteiden lopputulosta odotusarvoihin ja niiden toimivuutta on testattu CNC-koneen ajojärjestelmässä.

5.3 Integrointi

5.3.1 PLT-tiedostojen integrointi

Sovelluksen lukemat PLT-tiedostot on toteutettu suurimmaksi osaksi FlexiSIGN -mallinnusohjelmalla. Datan lukemista varten tiedoston näkymättömät merkit on purettu näkyviksi. PLT-tiedoston integrointi on toteutettu oman datakäsittelijän kautta. Datakäsittelijän tarkoituksena on lukea raaka data ja muodostaa siitä objektirakenne. Datakäsittelijä säilyttää tiedoston järjestyksen indeksoimalla kaikki objektit ja niiden sisältämät pisteobjektit. Pisteobjekteihin liitetään niille kuuluvat merkinnät. Tällöin kuva on tallennettuna aliobjekteina ja niiden sisältäminä pisteobjekteina. Kuva sisältää satunnaisen määrän aliobjekteja ja aliobjektit voivat sisältää satunnaisen määrän pisteitä. Aliobjektien määrittelyä voisi kuvailla visuaalisena elementtinä, joka alkaa kynän osuessa paperiin ja loppuu kynän noustessa paperista. (Ks. kuvio 18.)



Kuvio 18. Tiedostosta muodostettu rakennekaavio

Keskiössä ovat alimman tason pisteobjektit, joista kuva rakentuu. Pisteobjekti eroaa tavallisesta koordinaatistopisteestä siten, että se osaa koordinaattien lisäksi tiedostaa oman indeksinsä kuvassa ja merkata mahdolliset toiminnallisuudet. Vaikka aliobjektit koostuvatkin pisteistä, ne on rakennettu selkeyttämään rakennetta ja helpottamaan järjestysmuutoksia.

5.3.2 Työtiedostojen integrointi ajojärjestelmään

PLT-tiedoston tallentamisen toiminta on hyvin pitkälti käänteistä tiedoston lukemiselle. Datan lukemisvaiheessa kaikki aliobjektit on indeksoitu ja niiden sisältämät pisteobjektit myös indeksoidaan. Indeksointi mahdollistaa tallentamisen käänteisessä järjestyksessä. Huomioitavaa on tiedoston muutokset, jotka on syytä validoida tallentamisen yhteydessä. Aliobjekteja ei tarvitse indeksiä lukuun ottamatta validoida, mutta pisteet, jotka aliobjekti sisältää, on syytä tarkastella tarkemmin. Tällöin vältytään virheellisiltä merkinnöiltä ja koordinaattien puutoksilta. Tallennuksen yhteydessä kaikki piilotetut merkit palautetaan takaisin omille paikoilleen, joita ajo-ohjelma sitten lukee. (Ks. kuvio 19.)

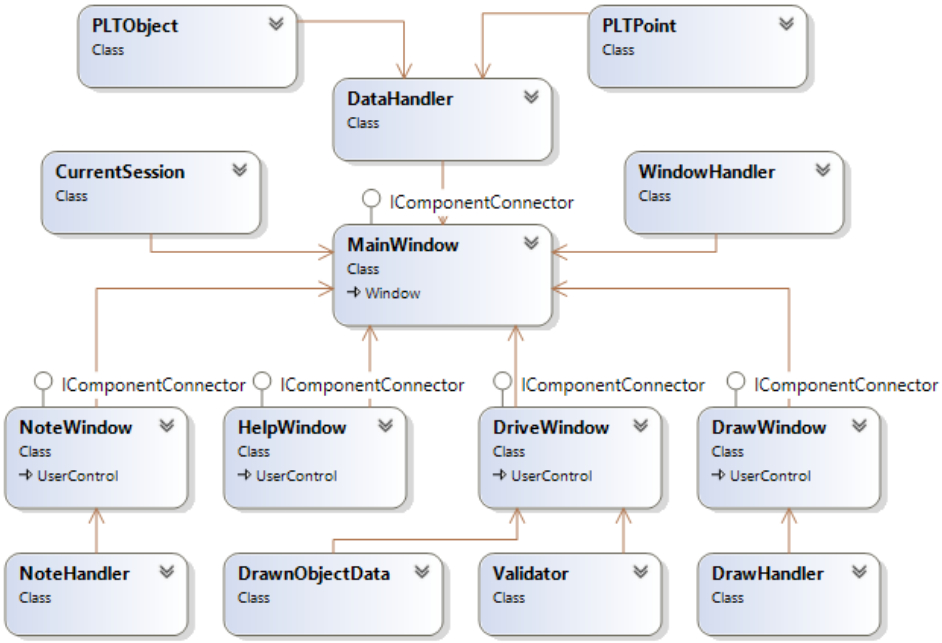
```
228             SaveFileDialog sfd = new SaveFileDialog();
229             sfd.Filter = "PLT File | *.plt";
230             //Opens filedialog to save file
231             try
232             {
233                 bool? result = sfd.ShowDialog();
234                 if (result == true)
235                 {
236                     string path = System.IO.Path.GetFullPath(sfd.FileName);
237                     if (sfd.CheckPathExists)
238                     {
239                         File.WriteAllText(path, GeneratePLTString(drawnCoordinatePoints));
240                     }
241                 }
242             }
```

Kuvio 19. PLT-tiedostoon tallentaminen

6 Tulokset

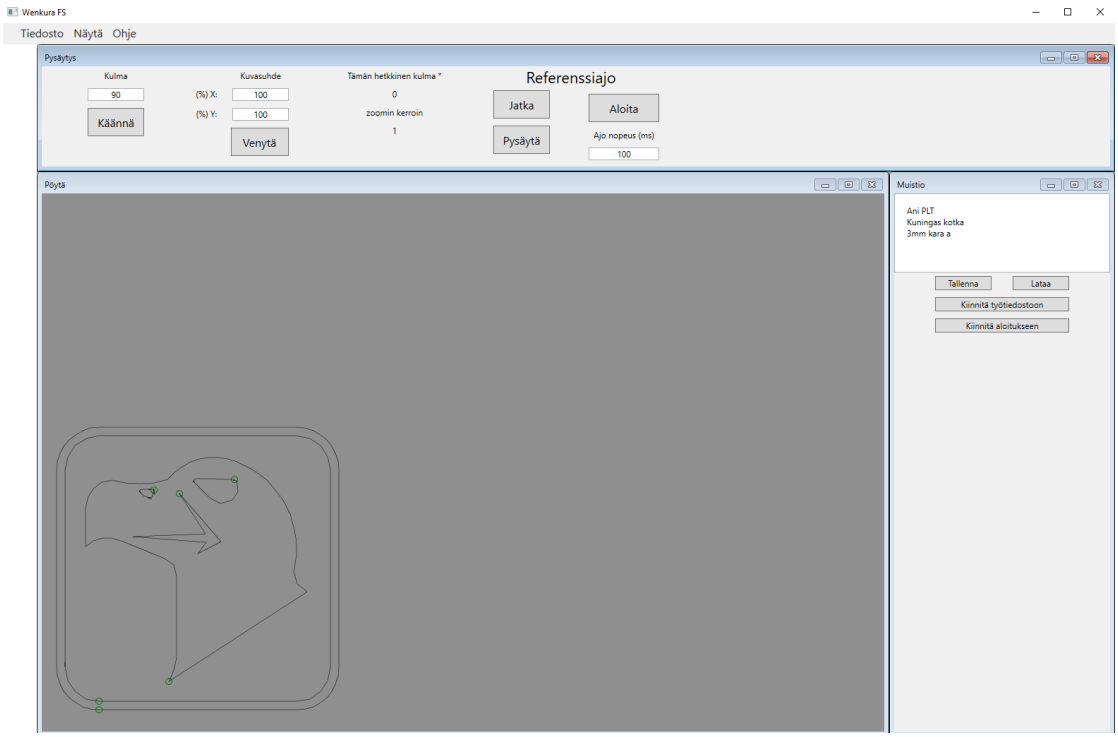
Sovelluksen suunnitteluvaiheessa toteutettu luokkakaavio osui lähelle sovelluksen todellista rakennetta. Suunnitellussa luokkakaaviossa runko on pysynyt samana kuin toteutuneessa, mutta tarkentavia yksityiskohtia on tullut lisää. Suurimpana muutoksena on rakenteeseen tullut piirretyn datan luokka. Tämä luokka toimii

tiedostosta luetun datan ja muokkaamisen välikätenä. Kyseinen luokka helpottaa monien toiminnallisuuksien toteuttamista, kuten muokkauksen palautusta ja tallennusta. Mielestäni luokkakaavion suunnittelu onnistui hyvin, sillä suunniteltua arkkitehtuuria tarvitsi vain tarkentaa. (Ks. kuvio 20.)



Kuvio 20. Sovelluksen toteutunut rakenne

Sovellus on suunnitelman mukaisesti rakennettu päänäkymän sisään, jossa jaottelu pienempiin näkymiin on toteutettu käyttäjätarinoiden pohjalta. Käyttöliittymän toteutuksessa on tehty pikakomentoja helpottamaan ja nopeuttamaan sovelluksen käyttöä. Suunnitelmassa mainittua jatkokehitystä on jo huomioitu siten, että sovelluksen sisällä on oma ikkunanhallintajärjestelmä. Sovellus onnistui käytettävyyden ja selkeyden puolesta hyvin, mutta palvelun ulkoasussa olisi vielä parannettavaa. Pieni asettelu, muotoilu ja paremmat indikaatiot parantaisivat käyttöliittymän selkeyttä ja käytettävyyttä. Kuitenkin toteutuneessa käyttöliittymässä on menty huomattavasti eteenpäin hahmotelmaan nähden. (Ks. kuvio 21.)



Kuvio 21. Toteutunut Käyttöliittymä

Toteutettujen toiminnallisuuksien testaaminen onnistui erinomaisesti, ja sen takia ensimmäinen tuotantoversio onkin jo julkaistu. Pääsääntöisesti testaus toteutettiin suunnitelman mukaisesti MSTestv2 Frameworkin avulla. Testit on toteutettu luokkakohtaisiin testisalkkuihin, tätä ei ole huomioitu sovelluksen testauksen suunnitteluvaiheessa. Selkeyden vuoksi testien jakaminen osiin on hyvä tapa. Testit ajetaan automaattisesti sovelluksen kääntövaiheessa. Testausiterointi sujui ongelmitta, ja toiminnallisuudet ovat hyväksyntätestattuja toimeksiantajan toimesta.

Toiminnallisuudet muodostuivat käyttäjätarinoiden pohjalta. Käyttäjätarinat kuvasivat selvästi haluttuja toiminnallisuuksia, joista kaikki toteutuivat. Käyttäjätarinoiden lisäksi jatkokehitystä ajatellen sovellukseen toteutettiin oma ikkunanhallintajärjestelmä, joka helpottaa uusien näkymien lisäämistä ja hallinnointia. Toiminnallisuudet jaettiin kaavaillusti näkymien kesken, eikä muutoksia suunnitelmaan tullut. Toiminnallisuuksia ei rakennekaaviota tarkemmin suunniteltu.

7 Pohdinta

7.1 Lokalisointi Windows 7:n ja 10:n välillä

Windows 7:ssä ja Windows 10:ssä sovelluksen oletuskieli asetetaan eri tavalla. Tällöin Windows 10:ssä käytetty lokalisointi ei toiminut Windows 7:ssä. Tämä aiheutti ongelman datan tallennuksessa, kun Windows 7 tallensi desimaaliluvut oletuksena pilkun kanssa ja Windows 10 tallensi oletuksena pisteen kanssa. Korjauksena toimi sovelluksen lokalisointi käyttöjärjestelmän mukaan.

7.2 PLT-tiedostosta muodostettu objektirakenne

PLT-tiedoston datasta luotiin aluksi yksi objekti, joka sisälsi kaikki kuvan pistekoordinaatit. Muokkaustoiminnallisuuksia toteuttaessa objektin rakenne aiheutti ongelmaa luetun datan käsittelyssä. Kuvioiden ajojärjestystä muokattaessa selvisi muokkaamisen olevan haasteellista kyseisellä datarakenteella.

Ajojärjestyksen muokkaamisen helpottamiseksi jokaisesta kuvassa olevasta kuviosta muodostetaan aliobjekti. Tällöin erillisten kuvioiden pistekoordinaatteja pystytään helposti muokkaamaan sotkematta datan rakennetta ja varmentamaan haluttu ajojärjestys.

7.3 Referenssiajo

PLT-tiedoston datasta muodostetun kuvan avulla pystytään simuloimaan CNC-koneen ajoa ja ajojärjestystä. Referenssiajo mahdollistaa niin ajonaikaiset tarkistukset, nopeuden säätämisen kuin pysäytyksetkin. Referenssiajon avulla muokatusta tiedostosta voidaan tarkistaa sen ominaisuudet ja virheet hetkessä. Referenssiajon onnistumisen takana on piirtämisen hoitava erillinen säie. Piirtäminen tapahtuu tässä säikeessä, jolloin kuvio on jatkuvasti muokattavissa.

7.4 Automatisointi

Luetusta datasta automatisoidaan tavallisesti tehtävät manuaaliset toiminnot, kuten kuvion asettelu ja sijainnin määrittely. Automatisointia voidaan säätää erillisestä

konfiguraatiotiedostosta. Tämä mahdollistaa automatisoinnin muokkauksen ja tarvittaessa sen kytkemisen pois päältä. Erilaisten konfiguraatioiden lisäksi PLT-tiedostoihin voidaan liittää muistiinpanot, jotka luetaan, kun PLT-tiedosto avataan sovelluksessa.

7.5 Teknologiavalinnat

Erityisesti WPF:n käyttö kyseiseen toteutukseen oli erinomainen teknologiavalinta. WPF:n tarjoamat elementit mahdollistivat nopean käyttöliittymän luomisen. WPF MDI DLL:n avulla ominaisuuksien jaottelun selkeisiin näkymiin tapahtui ongelmitta. WPF tarjoaa 2D- ja 3D grafiikoiden piirtämiseen valmiita toiminnallisuuksia, joita pystyttiin hyödyntämään sovellusta kehittäessä.

8 Jatkokehitys

8.1 Lisäautomatisointi

Sovelluksen myötä on huomattu uusia automatisoitavia asioita. Suurimmaksi puheenaiheeksi nousi tiedostosta ylimääräisten pisteiden automaattinen poistaminen. Poistaminen kohdistuisi tuplapisteisiin sekä irrallisiin pisteisiin. Pisteiden poistamisen lisäksi työstönopeutta terävien kulmien kohdalla voisi hidastaa, joka parantaisi niiden leikkausjälkeä.

8.2 Ajo-ohjelman yhdistäminen hallinnointijärjestelmään

Ajo-ohjelman siirtäminen hallinnointijärjestelmään on täysin erillinen projekti, joka on huomioitu suunnitteluvaiheessa. Itse PCI-7334 ajokorttia ohjaava DLL on käytettävissä suoraan wrapperin avulla. Kuitenkin suurin työ näiden ohjelmien yhdistämisessä olisi ajo-ohjelman testaaminen. Testaus tulisi toteuttaa monella eri tavalla, kuten yksikkötesteillä, automaatiotesteillä sekä käyttäjätesteillä. Lisäksi testeissä käytettävän data määrän kasaaminen vaatisi paljon työtä.

8.3 Piirtopöytää isompien kuvien tallentaminen osissa

CNC-koneen fyysistä työstöpöytää isommat kuvat ovat harvinainen tapaus. Tällaisessa tapauksessa olisi hyödyllistä pystyä merkitsemään koordinaatisto piirtoalueelle. Tämän jälkeen piirtopöydälle tulisi toteuttaa raahaus ominaisuus, jolloin kuvan voisi asettaa keskelle origoa. Sovelluksella tulisi pystyä jakaamaan tiedosto osiin, joista jokainen osa voisi olla esimerkiksi yksi koordinaatiston sektori. Tällöin suurienkin kuvioiden toteuttaminen olisi huomattavasti helpompaa.

Lähteet

Allain, A. 2018. Why Should You Learn C?. Verkkosivusto. Viitattu 18.1.2018. https://www.cprogramming.com/whyc.html

Blome, M., Jones, M., Peek, B., Robertson, C. & Waren, G. 2016. Introduction to WPF. Verkkosivusto. Viitattu 21.1.2018. https://docs.microsoft.com/en-us/visualstudio/designers/introduction-to-wpf

Blome, M., Latham, L., Levin, I., Wagner, B. & Wenzel, M. 2015. Introduction to the C# Language and the .NET Framework. Verkkosivusto. Viitattu 16.1.2018. https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

Blome, M., Latham, L., Pratt, T. & Wenzel, M. 2015. Visual Basic Language Features. Verkkosivusto. Viitattu 16.1.2018. https://docs.microsoft.com/en-us/dotnet/visual-basic/programming-guide/language-features/index

Branching and Merging. 2018. Verkkosivusto. Viitattu 21.1.2018. https://git-scm.com/about

Brown, K., Jones, M., Lee, T., Robertson, C & Warren, G. 2017. Visual Stuide IDE overview. Verkkosivusto. Viitattu 28.1.2018. https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-ide

Chapter 3: Architectural Patterns and Styles. 2018. Verkkosivusto. Viitattu 26.1.2018. https://msdn.microsoft.com/en-us/library/ee658117.aspx

Code Analysis for Managed Code Overview. 2018. Verkkosivusto. Viitattu 28.1.2018. https://msdn.microsoft.com/en-us/library/3z0aeatx.aspx

Git Workflow. 2018. Verkkosivusto. Viitattu 4.3.2018. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Gray Box Testing: Process, Techniques, Strategy, Challenges. 2018. Verkkosivusto. Viitattu 26.1.2018. https://www.guru99.com/grey-box-testing.html

How to design a great user experience for desktop applications. 2018. Verkkosivusto. Viitattu 25.1.2018. https://msdn.microsoft.com/en-us/library/windows/desktop/dn742462(v=vs.85).aspx

Jones, M., Latham, L., Petrusha, R., Pratt, T. & Wenzel, M. 2017. Overview of the.NET Framework. Verkkosivusto. Viitattu 21.1.2018. https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview

Nissinen, J. 2018. Toimitusjohtaja. Yes Neon Oy. Haastattelu 22.1.2018.

Parikshit, P. 2014. Three Layer Architecture in C# .NET. Verkkosivusto. Viitattu 28.1.2018. https://www.codeproject.com/Articles/36847/Three-Layer-Architecture-in-C-NET

What is BLACK Box Testing? Techniques, Example & Types. 2018. Verkkosivusto. Viitattu 26.1.2018. https://www.guru99.com/black-box-testing.html

What is WHITE Box Testing? Techniques, Example & Types. 2018. Verkkosivusto. Viitattu 26.1.2018. https://www.guru99.com/white-box-testing.html

Liitteet

Liite 1. Kuvion piirtäminen

```
408         for (int i = index; i < drawnCoordinatePoints.Count; i++)
409         {
410             Line lineToNextPoint = new Line();
411             lineToNextPoint.Stroke = System.Windows.Media.Brushes.Black;
412             lineToNextPoint.StrokeThickness = 0.5;
413 
414             if (drawnCoordinatePoints[i].isDrawAble)
415             {
416                 if (drawnCoordinatePoints[i - 1].shortCode == "PU")
417                 {
418                     lineToNextPoint.X1 = drawnCoordinatePoints[i].x;
419                     lineToNextPoint.Y1 = drawnCoordinatePoints[i].y;
420                 }
421                 else if (double.IsNaN(tmpX) || double.IsNaN(tmpY))
422                 {
423                     lineToNextPoint.X1 = drawnCoordinatePoints[i].x;
424                     lineToNextPoint.Y1 = drawnCoordinatePoints[i].y;
425                 }
426                 else
427                 {
428                     lineToNextPoint.X1 = tmpX;
429                     lineToNextPoint.Y1 = tmpY;
430                 }
431                 tmpX = lineToNextPoint.X2 = drawnCoordinatePoints[i].x;
432                 tmpY = lineToNextPoint.Y2 = drawnCoordinatePoints[i].y;
433 
434                 if ((bool)useReference)
435                 {
436                     if (paused == false)
437                     {
438                         await PutTaskDelay(lineToNextPoint, (int)driveSpeed, canvas);
439                     }
440                     if (paused == true)
441                     {
442                         pausedIndex = i;
443                         break;
444                     }
445                     if (i + 1 == drawnCoordinatePoints.Count())
446                     {
447                         drivingReference = false;
448                     }
449                 }
450                 else
451                 {
452                     canvas.Children.Add(lineToNextPoint);
453                 }
454             }
455         }
```