

Jasmiina Heikkilä

PLC:N OHJAINOHJELMAN SUUNNITTELU JA TUOTTAMINEN
ASIAKKAALLE

Tietojenkäsittelyn koulutusohjelma
2016

PLC:N OHJAINOHJELMAN SUUNNITTELU JA TUOTTAMINEN ASIAKKAALLE

Heikkilä, Jasmiina
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Lokakuu 2016
Ohjaaja: Hentunen, Ilmari
Sivumäärä: 42
Liitteitä: 1

Asiasanat: ohjelmistosuunnittelu, ohjelmistotuotanto, ohjelmoitavat logiikat, sarjaportit

Tämä opinnäytetyö käsittelee ohjainohjelman kehittämistä asiakkaalle tietyn PLC-laitteen käyttämiseksi ja valvomiseksi. Ohjainohjelman tehtävänä oli korvata edellinen käytössä ollut ohjelma sekä täyttää asiakkaan toiveet ja tarpeet. Työ toteutui pienenä ohjelmistotuotantoprojektina, joka on osa asiakkaan suurempaa uudistusprojektia.

Opinnäytetyössä käydään läpi projektin kulku sen alkuvaiheista ja suunnittelusta aina ohjelman valmistumiseen. Sekä projektisuunnittelun että ohjelman kehittämisen teoreettinen ja käytännöllinen osuus kuvataan vaiheittain. Itse tutkimusongelma on asiakkaalle sopivan ohjelmistotuotteen suunnittelu ja toteuttaminen.

Ensimmäisenä aiheena käsitellään koko projektin lähtökohdat, joista tärkein kokonaisuus on vaatimusmäärittely. Projektisuunnitteluosiossa kerrotaan teorian ohella lopulliset ratkaisut ohjelmistotuotantoprojektin toteuttamiseen. Ohjelmistosuunnitteluosio kattaa teorian, suunnittelun ja ohjelman toteuttamisen käytännössä. Läpi käydään projektissa käytetyt ohjelmat ja tekniikat, arkkitehtuurisuunnittelu sekä luokkasuunnittelu, käyttöliittymä ja ohjainohjelman toiminnallisuus.

DESIGNING AND ENGINEERING OF A PLC CONTROL PROGRAM FOR A CUSTOMER

Heikkilä, Jasmiina

Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences

Degree Programme in Information and Communication Technology

Month 2016

Supervisor: Hentunen, Ilmari

Number of pages: 42

Appendices: 1

Keywords: software design, software engineering, programmable logics, serial ports

This thesis addresses the development of a control program for a client for the purpose of using and monitoring a specific PLC. The purpose of the control program was to replace the previous program in the client's use as well as to fulfill their wishes and needs. The task was actualized as a small software production project, which is a part of the client's larger-scale renovation project.

The thesis undergoes the project's progression from its beginning phases and planning all the way to the completing of the program. The theoretical and functional aspects of both the project planning and the development of the program are demonstrated in stages. The research problem itself is the designing and engineering of a suitable software product for the client.

The first addressed subjects are the bases of the entire project, the most important aspect of which is requirements specification. Theory along with definitive decisions concerning the execution of the software production project are explained in the project planning part. The software design part covers the theory, the design and the execution of the program in practice. The software and technology used in the project, architecture design as well as class design, user interface and the functionality of the control program will be gone over.

SISÄLLYS

1	JOHDANTO.....	
2	PROJEKTIN LÄHTÖKOHDAT.....	
2.1	Lähtökohtien kuvaus.....	
2.2	Vaatusmäärittely.....	
2.3	Riskien ja ongelmien kartoitus.....	
3	PROJEKTISUUNNITTELU.....	
3.1	Yleistä projekteista.....	
3.2	Aikataulu.....	
3.3	Dokumentointi.....	
3.4	Laadunhallinta.....	
3.5	Tuotteenhallinta.....	
3.6	Testaussuunnittelu.....	
4	OHJELMAN SUUNNITTELU.....	
4.1	Teknologia.....	
4.1.1	Ohjelmat ja tekniikat.....	
4.1.2	Siemensin 3964R-protokolla.....	
4.2	Datapaketeista.....	
4.3	Arkkitehtuurisuunnittelu.....	
5	TOTEUTUS JA TOIMINNALLISUUS.....	
5.1	Luokat.....	
5.1.1	Form1.Designer.....	
5.1.2	Form1	27
5.1.3	Data	28
5.1.4	SP	30
5.2	Käyttöliittymä.....	
5.3	Ohjelman toiminnallisuus.....	
5.3.1	Yleistä	34
5.3.2	Tuotanto- eli perusnäkö.....	
5.3.3	Tilausnäkö	37
5.3.4	Asetusnäkö	37
5.3.5	Käsiäjonäkö.....	
6	YHTEENVETO.....	
	LÄHTEET.....	
	LIITTEET	

1 JOHDANTO

Opinnäytetyöni aiheena oli suunnitella ja tuottaa ohjainohjelma yhdelle asiakasyrityksen käyttämistä PLC-laitteista. Työ toteutettiin pienenä ohjelmistotuotantoprojektina, jossa toimittajaa edustin minä. Toimin tiiviissä yhteistyössä asiakkaani kanssa, mutta vastuu projektin johtamisesta ja toteuttamisesta oli yksin minun harteillani.

PLC (programmable logic controller) tarkoittaa ohjelmoitavaa logiikkaa. PLC:n ohjainohjelma tarkoittaa sellaista ohjelmaa, jolla PLC-laitteelle annetaan ohjeita ja sen toimintaa sekä tilaa seurataan. Asiakasyritykselläni kaikissa suurissa tehdaskoneissa on sisällä ohjelmoitu logiikka, joita työntekijät hallinnoivat koneisiin sarjaporttien avulla kytkettyjen tietokoneiden kautta. Oma projektini oli osa suurempaa uudistusprojektia, jossa pikku hiljaa korvataan vanhoja ohjainohjelmia sekä tulevaisuudessa otetaan käyttöön tehdaskoneiden paikallisverkko.

Opinnäytetyössäni käydään vaiheittain läpi projektin eteneminen alusta loppuun. Käsittelen sekä lähtökohtien hahmottelua, vaatimusmäärittelyä ja projektisuunnittelua että ohjelmistosuunnittelua ja lopputuotteen toteuttamista. Esittelen ohjainohjelman taustateoriaa lyhyesti sekä kuvaan ohjelman toiminnallisuutta käyttöliittymän avulla.

2 PROJEKTIN LÄHTÖKOHDAT

2.1 Lähtökohtien kuvaus

Ohjelman elinkaaren alkuun kuuluu yleensä esitutkimus, jota seuraa vaatimusmäärittely. Vaiheiden tärkein tarkoitus on selvittää se tarve, jonka täyttämiseksi uusi ohjelma kehitetään. (Pohjonen 2002, 26.) Asiakkaani oli Versowood Oy:n alla toimiva Porin puulavatehdas. Projektissa toimin tiiviissä yhteistyössä ns. asiakasedustajani kanssa, joka oli tehtaan tuotantopäällikkö. Aloitin työni haastattelemalla häntä kartoittaakseni sekä ohjelman lähtökohdat ja vaatimukset.

Asiakkaan tehtaalla käytössä olevat PLC:t on ohjelmoitu lähemmäs 20 vuotta sitten. Viime vuosina tarve järjestelmien uusimiselle on kuitenkin kasvanut teknologian kehittymisen vuoksi. Ohjainohjelmat ovat yhtä vanhoja kuin PLC:tkin, lisäksi ne ovat kaikki yhden ja saman henkilön suunnitteleamia ja ohjelmoimia. Minun tehtävänäni oli kehittää uusi ohjainohjelma jalaskoneelle, joka sahaa jalaksia ja naulaa ne kiinni puulavoihin. Sain juuri tämän ohjelman kontolleni on siitä syystä, että edellinen toimi kaikista tehtaan ohjainohjelmista huonoiten. Se oli kymmenisen vuotta sitten hätäisesti kokoon kyhätty vajavainen korvike, sillä alkuperäinen ohjainohjelma kadotettiin tietokoneikon yhteydessä. (Heikkilä, henkilökohtainen haastattelu 13.5.2016.) Asiakkaan tarve oli saada uusi ohjelma, joka olisi toimiva ja hyvä käyttää.

Työni toteutettiin pienenä ohjelmistotuotantoprojektina. Haikala ja Mikkonen (Ohjelmistotuotannon käytännöt, 11) määrittelevät sanan näin: ”Käsite software engineering, suomeksi ohjelmistotuotanto tai -tekniikka, tarkoittaa tietokoneohjelmistojen rakentamisessa yleisesti käytettyjä tekniikoita, työkaluja, menettelytapoja ja periaatteita. Termi ohjelmisto puolestaan kattaa sekä tietokoneohjelman että siihen liittyvän dokumentaation.”

Opinnäytetyön tutkimusongelma on itse ohjainohjelma. Käytännössä tämä tarkoittaa, että asiakkaan tarvitsema uusi ohjelma on ns. pulma, jota minä tutkin ja pyrin ratkomaan. Vaatimusmäärittely muodostaa kriteerit ongelman ratkaisemiselle. Suunnittelu on ratkaisun väline ja toteutus on itse ratkaisu, jota asiakkaalle tarjotaan. Tutkimusta tapahtuu joka vaiheessa ja sen tavoitteena on tuottaa asiakkaalle mahdollisimman hyvä ratkaisu eli ohjainohjelma. Projekti tarjoaa puitteet ongelmaan perehtymiselle ja sen parissa työskentelemiselle. Projektisuunnittelun tavoitteena on yksinkertaisesti varmistaa, että nämä puitteet ovat mahdollisimman sopivat.

2.2 Vaatimusmäärittely

Ohjelman toteutumisen onnistuneisuuden arviointi perustuu sille asetettuihin määrittelyihin. Määrittelyt ovat asiakkaan tarpeita ja haluja, jotka luovat pohjan koko ohjelman suunnittelulle ja toteutukselle. Vaatimusmäärittely itsessään tarkoittaa yleensä dokumenttia, johon kerätään nämä ohjelmalta tarvittavat ominaisuudet (ei-toiminnalliset vaatimukset) ja toiminnot (toiminnalliset vaatimukset) sekä reunaehdot (ohjelman toteuttamisen tekniset raamit). Vaatimusten kerääminen ja määrittely vaatii toimittajan ja asiakkaan tiivistä yhteistyötä, mikä puolestaan edellyttää kommunikointiin panostamista. Yleisiä ongelmia ovat vaatimusten tulkinnan vaikeus, epäselvyys, mahdollinen keskinäinen ristiriitaisuus ja määrän tai tarkkuuden riittämättömyys. Lisäksi lähes aina joudutaan tekemään kompromisseja reuna-ehdojen perusteella. (Pohjonen 2002, 28 – 30, 34; Suomen Automaatioseura ry 2005, 74 - 75, 97 – 98.)

Virheettömyyden ohella hyvän vaatimuksen on oltava jollakin tavalla mitattavissa, jotta sen toteutuminen pystytään projektissa osoittamaan. Tähän liittyy läheisesti vaatimuksen jäljitettävyyden, mikä tarkoittaa alkuperäisen vaatimuksen tunnistamista käytännön ratkaisusta. Vaatimukset tulee myös kuvata riittävän tarkasti, mutta toisaalta ei tulisi asettaa toteutukselle turhia rajoitteita. (Haikala & Mikkonen 2011, 63 - 64; Suomen Automaatioseura ry 2005, 99.) Mielestäni tärkeimpien vaatimusten tunnistaminen ja kuvaaminen muita tarkemmin oli hyvä ajatus.

Toteutin vaatimusmäärittelyn itsenäisesti, mutta se perustui asiakasedustajani haastatteluun (Heikkilä, 13.5.2016). Kirjoitin silloin ylös kaikki edellisen ohjainohjelman ongelmat, joita analysoimalla muodostin ratkaisuehdotuksista vaatimusmäärittelyn. Esittelin sen sitten asiakasedustajalleni, joka hyväksyi sen pienin korjauksin. Tästä dokumentista tiivistetty taulukko on opinnäytetyössä liitteenä (Liite 1).

Ohjainohjelmalla oli tiukat reunaehdot, sillä sen oli toimittava yhteen valmiin PLC:n kanssa. Lisäksi minun oli perehdyttävä sen käyttämään sarjaporttiteknoologiaan. Ohjelmaa käytetään Windows 7 -käyttöjärjestelmällä. Käyttöliittymäsuunnittelua ohjaavaksi avainsanaksi nostettiin käytettävyys, joka tarkoittaa sitä, kuinka helposti ja sujuvasti käyttäjä kykenee saavuttamaan päämääränsä ohjelmalla (Kuutti 2003, 13, 47). Syynä tähän oli edellisen ohjainohjelman käytön vaikeus ja epämukavuus (Heikkilä, henkilökohtainen haastattelu 13.5.2016). Hyvään käytettävyyteen kuuluvat mm. käytön miellyttävyys, helppous, tehokkuus ja opittavuus (Kuutti 2003, 13, 47). Itse koin terveen järjen parhaaksi keinoksi arvioida käytettävyyttä. Lisäksi hyödynsin projektin aikana lähes kaikki mahdollisuuteni saada palautetta asiakkaalta käytettävyyden laadun varmistamiseksi.

Vaatimusmäärittelyssä ei tule unohtaa mahdollisuutta, että vaatimukset muuttuvat projektin aikana. Muutoksia ei pidä kuitenkaan koskaan tehdä tällaisiin projektin peruspilareihin ilman kunnollista syytä ja johtoryhmän tms. hyväksyntää. Vaatimusten hallinnan tehtävänä on huolehtia vaatimusmäärittelystä, erityisesti sen muutoksista. (Haikala & Mikkonen 2011, 67.) En kokenut tarpeelliseksi ryhtyä mihinkään erillisiin toimiin vaatimusten hallitsemiseksi. Määrittelyn ydinkohtiin ei onneksi tullut mitään muutostarpeita, lähinnä vain käyttöliittymää ja toiminnallisuuden hienouksia koskeviin asioihin.

2.3 Riskien ja ongelmien kartoitus

Riski tarkoittaa mitä tahansa, mikä projektissa voi muodostua ongelmaksi. Riskienhallintaan riittävät yleensä yksinkertaiset keinot, vaikka kokonaisuus voi vaikuttaa monimutkaiselta. (Lehtimäki 2006, 79.) Tässä projektissa ei ollut käytössä

minkäänlaista varsinaista riskienhallintametodia, mutta keskustelin useaan otteeseen asiakasedustajan kanssa mahdollisista riskeistä ja ongelmista. Pidin tätä riittävänä toimintana projektin pieneen kokoon nähden.

Suurin ongelma koko työssä oli se, että lähes kaikki saatavilla oleva tieto PLC:n sekä edellisen ohjainohjelman toiminnasta oli itse tässä ohjainohjelmassa. Sain käyttööni sen ohjelmakoodin sekä sen että PLC:n alkuperäisen suunnittelijan ja ohjelmoijan sähköpostiosoitteen. Koodi oli selvästi vanhaa, osin hyvin monimutkaista ja kirjoitettu minulle ennestään vieraalla Delphillä (Jalasta 2005). Kyseessä on yhdeksänkymmentäluvulla kehitetty Visual Basicin kilpailija, joka tarjoaa mahdollisuuden visuaaliseen sovelluskehitykseen (Järvinen & Piispa 2000, 3). Jo heti projektin alussa oli siis selvää, että tiedonkeruu suunnittelua ja toteutusta varten ei tule olemaan helppoa. Pahimmillaan tästä voisi seurata, että kokonaisia toiminnallisuuksia ei voida ollenkaan toteuttaa.

Toinen merkittävä ongelma on tietysti projektin epäonnistumisen riski. Yleisin syy tähän on projektihallinnassa, mitä ohjelmistoprojektien kasvaminen aina vain suuremmiksi ja monimutkaisemmiksi vaikeuttaa entisestään. (Pohjonen 2002, 17.) Tämän projektin pienen koon vuoksi en kuitenkaan ollut liian huolissani. Uskoin järjen käytön sekä hyvän yhteistyön asiakasedustajan kanssa auttavan tarkkoja kirjallisia suunnitelmia enemmän. Muita mahdollisia riskejä olivat virheet sekä yllättävät tai suuret muutostarpeet. Molemmat johtaisivat siihen, että ohjelmaan voidaan joutua tekemään mittaviakin korjauksia. Pyrin huomioimaan nämä riisit välttämällä liiallista tiukkuutta vaatimusmäärittelyssä. Ohjelmistosuunnittelun selkeydestä huolehtiminen vaikutti myös hyvältä keinolta varautua mahdollisiin muutos- tai korjaustarpeisiin.

3 PROJEKTISUUNNITTELU

3.1 Yleistä projekteista

Niin pientä projektia ei ole olemassakaan, etteikö siitä kannattaisi tehdä minkäänlaista suunnitelmaa. Projektisuunnitelmaa tehtäessä voidaan aiheuttaa merkittävästi vahinkoa liiallisella välinpitämättömyydellä tai liiallisella monimutkaisuudella. (Haikala & Mikkonen 2011, 167.) Tämä projekti oli kuitenkin niin pieni ja kevytrakenteinen, että projektisuunnitelma koostui suoraviivaisesti tekstitiedostosta ja keskusteluista asiakasedustajan kanssa. Myös projektiseuranta toteutui lähes päivittäisillä keskusteluilla (Lehtimäki 2006, 14).

Ohjelmistotuotannon perinteinen projektimalli on vesiputousmalli, jossa eri vaiheet seuraavat toisiaan kiinteästi järjestyksessä aina määrittelystä käyttöönottoon ja ylläpitoon. Kyseessä on lähinnä ideaali, joka löytyy yleisesti ottaen kaikkien uudempien projektimallien sisältä muodossa tai toisessa. Iteroiva ohjelmistokehittäminen on käytännöllisempää, mutta vaikeampaa hallita. Kaikkein yksinkertaisin lähestymistapa on ihan vain ohjelman kasvattaminen ja kehittäminen ilman muuta suunnitelmaa kuin työstää sitä, kunnes se on valmis. (Haikala & Mikkonen 2011, 29 - 30, 37; Lehtimäki 2006, 153; Pohjonen 2002, 40.) Koin vesiputousmallin löyhän noudattamisen järkeväksi ja systemaattiseksi tavaksi organisoida ohjelmistotuotantoprojekti ja edetä siinä. Itse ohjainohjelman toteuttamiseen suhtauduin kuitenkin omana kokonaisuutenaan, jossa koodaaminen ja testaaminen tapahtuivat iteroivasti.

Projektille on aina määriteltävä laajuus tarkkaan, sillä projekti ilman tunnettua laajuutta ei edes kunnolla ole projekti. Kaikkien sidosryhmien tulee olla yhteisymmärryksessä siitä, mitä projektiin kuuluu. Laajuuteen ei myöskään pitäisi tehdä muutoksia ilman tarkkaa perustelua, läpikäyntiä ja ennen kaikkea ihmisille muutoksista informoimista. (Lehtimäki 2006, 47 - 48, 52.) Tämän projektin laajuus oli alusta alkaen selvä. Tehtävänäni oli toteuttaa ohjainohjelma vaatimusten

mukaisesti kohtuullisessa aikataulussa, minkä jälkeen projekti päättyy. Tämän jälkeiset ylläpitoon liittyvät toimet ja vaatimusmäärittelyssä esiin nostetut tulevaisuuden muutostarpeet rajautuvat kaikki projektin laajuuden ulkopuolelle. Laajuuteen liittyä, että projektille on olemassa hyväksymis- tai ainakin päättämiskriteerit (Lehtimäki 2006, 179). Asiakasedustaja asetti hyväksymiskriteeriksi ohjainohjelman virheettömän toiminnan tietyn ajanjakson ajan. Hyväksyin tämän päätöksen.

Ohjelmallani oli kourallinen käyttäjiä, joihin myös asiakasedustaja kuuluu. Asiantuntijatukena oli sekä PLC:iden että niiden ohjainohjelmien suunnittelija ja ohjelmoija. Muita sidosryhmiä en tunnistanut. Projektipäällikön eli minun toimintaani valvova ohjausryhmä tulisi olla (Lehtimäki 2006, 40), mutta rooli lankesi epävirallisesti asiakasedustajalleni. Hän oli projektin hierarkiassa korkeimmalla paikalla ja mm. hyväksyi kaikki suuret päätökset.

Projektipäällikkö on vastuussa siitä, että kaikilla projektiin sidoksissa olevilla henkilöillä on aina riittävästi paikkansapitävää tietoa. Suunnitelmia yms. voidaan esittää erilaisilla mallinnustekniikoilla, joista merkittävimpiä ovat UML-kaaviot. Tekniikoita valittaessa on kuitenkin huomioitava tiedon luonteen ohella henkilöt, joiden tulee tämä tieto ymmärtää. (Lehtimäki 2006, 164, 166 – 167.) Informoiminen itseni ja asiakasedustajan välillä tapahtui pääasiassa suullisesti. En nähnyt mitään syytä kuluttaa aikaa ns. hienojen kaavioiden tekemiseen, sillä lyijykynä ja paperi tai ohjainohjelman koodin ja toiminnallisuuden esittely riittivät aina havainnollistamaan asiani tälle yhdelle ihmiselle.

3.2 Aikataulu

Tämän ohjainohjelman käyttöönotolle ei ollut minkäänlaista palavaa kiirettä, sillä edellinen oli ongelmistaan huolimatta ollut käytössä jo useita vuosia. Aikataulun asettaminen jäi siis omille harteilleni. On myös mainittava, ettei minulle maksettu itse työstä, joten raha-asiat eivät vaikuttaneet aikataulun suunnittelemiseen. Asiakas osti minulta vasta valmiin ohjainohjelman.

Yleisesti ottaen projektin osittaminen on projektisuunnittelun tärkein asiakokonaisuus siitäkin huolimatta, että ohjelmistotyö on huonosti osittuvaa. Osittaminen tarkoittaa projektin jakamista pieniin, hallittaviin osiin, jotta ne voidaan asettaa yksittäisten ihmisten tehtäviksi. (Haikala & Mikkonen 2011, 157, 159.) Tässä projektissa tämä hyvin tärkeä kokonaisuus sivuutettiin täysin, sillä tehtävien osittaminen yhdelle hengelle vaikutti täysin turhalta. Arvioin vain omaa työtehokkuuttani, mitä kokemuksen puute projekteista hieman hankaloitti. Päätin mielummin varata työlle liikaa aikaa kuin liian vähän, mikä sopi myös asiakkaalleni.

Aikataulu muodostui lopulta vain muutamista kalenteriin asetetuista etapeista. Projektin päättymisen eli ohjainohjelman valmistumisen takarajaksi asetettiin vuoden 2016 jouluku. Käytännössä tämä tarkoitti noin kuutta kuukautta aikaa saattaa projekti alusta loppuun. Toinen tärkeä etappi oli asiakasedustajani kesälomalle lähteminen elokuun alussa, jonka tiesin etukäteen hidastavan työskentelyäni muutamaksi viikoksi. Pääsyy tähän oli se, etten yksin turvallisuussyistä saisi testata ohjainohjelmaa sen käyttöympäristössä tehdastiloissa. Asetin elokuun alkuun välietapin, joka oli saattaa ohjainohjelma mahdollisimman valmiiksi käytettävyydestä varten siihen mennessä. Kolmas ja ajallisesti ensimmäinen etappi oli päästä aloittamaan ohjainohjelman toteutus kesäkuun alettua. Koko projekti aloitettiin toukokuun alussa.

3.3 Dokumentointi

Ohjelmistoalaa vaivaava kiire johtaa yleensä siihen, ettei varsinkaan suunnittelusta ja määrittelystä tuoteta kunnollista dokumentaatiota. Toisaalta on olemassa myös riski, että dokumentoinnissa keskitytään epäoleellisiin asioihin ja tärkeä tieto hautautuu turhuuksien sekaan. Dokumentaation sopiva laajuus ja määrä riippuvat aina itse projektista; pienessä projektissa dokumentaation määrä on luonnollisesti vähäisempi kuin suuressa projektissa. (Pohjonen 2002, 79.)

Koko projektin dokumentointi oli yksin minun vastuullani. Asiakas ei tarvinnut minulta yhtikäs mitään dokumentaatiota, enkä itsekään kokenut tekeväni mitään

kaikella sillä kirjallisella tiedolla, joka lähdemateriaaleissa usein esitetään ehdottoman tärkeänä. Projektia koskevia asioista en kirjoittanut pahemmin ylös, sillä luotin muistiini. Toisaalta itse ohjainohjelman huolellinen dokumentointi oli nostettu esille sekä vaatimusmäärittelyssä että keskustelluissani asiakasedustajan kanssa. Tämä toteutui helposti ja luontevasti runsaalla ohjelmakoodin kommentoinnilla, joka auttoi myös saattamaan koodin luettavampaan ja selkeämpään muotoon. Kirjallista dokumentointia korvasin tietoisesti käyttämällä hyväkseni mahdollisuutta keskustella asiakasedustajan kanssa päivittäin. Pyrin pitämään hänet ajan tasalla projektin etenemisestä kertomalla ohjelman toiminnasta, ongelmista ja ratkaisuksista. Ajatuksena oli tutustuttaa asiakas ohjainohjelmaan mahdollisimman hyvin.

3.4 Laadunhallinta

Laatu itsessään on käsitteenä hankala. Ensinnäkin ohjelmistotuotantoprojektissa saatetaan tulkita laadun tarkoittavan joko lopputuotteen laatua, tuotteen tuotantoprosessin laatua tai asiakkaan saaman palvelun laatua. Toisin sanoen laadun voidaan ajatella tarkoittavan joko tuotteen tai palvelun kykyä vastata asiakkaan odotuksiin ja tarpeisiin. Projektissa keskitytään helposti vääriin asioihin, mikäli asiakkaan käsitys laadusta on epäselvä. Lisäksi on muistettava, että noudatetaan nimenomaan tätä asiakkaan käsitystä hyvästä laadusta. (Lehtimäki 2006, 66 - 69, 75; Pohjonen 2002, 78; Suomen Automaatioseura ry 2005, 112.)

Laadunhallinta tarkoittaa yleensä kaikkia keinoja sekä ohjelman virheiden määrän laskemiseksi että oikean laatutason varmistamiseksi. Laadunhallintajärjestelmä puolestaan tarkoittaa toimittajan toimintatapoja ohjelmistotuotantoprosessissa. (Haikala & Mikkonen 2011, 30, 139.) Itselläni ei tietenkään ollut olemassa minkäänlaista laadunhallintajärjestelmää. Pyrin siksi käyttämään tervettä järkeä sekä testaamaan ohjelmaa niin paljon kuin suinkin mahdollista. Toimintatapani rytmitin asiakasyrityksen mukaan.

Vaatimusmäärittelyn yhteydessä kävi selväksi, että asiakas tulkitsi laadun tarkoittavan tuotteen laatua. Hyvä laatu tarkoitti siis luonnollisesti ohjainohjelman kykyä täyttää sille asetetut vaatimukset. Ohjelman hyvä käytettävyyks oli asiakkaalle

erittäin tärkeää, joten sitä tuli painottaa suunnittelussa ja toteutuksessa. Sovimme laajasta käytettävyydestä asiakkaalle parhaan mahdollisen laadun aikaansaamiseksi. Toisaalta asiakas ei ottanut juurikaan kantaa ohjelmakoodin laatuun. Toimin kaikkia yleisiä ohjeistuksia vastaan tekemällä ns. turhaa työtä, kun pyrin kirjoittamaan omasta mielestäni hyvälaatuista ja siistiä koodia.

3.5 Tuotteenhallinta

Tuotteenhallinnan tehtävänä on huolehtia kaikista komponenteista, joista ohjelmisto koostuu. Merkittävin asiakokonaisuus on sekä tuotteen että sen komponenttien versionhallinta, johon liittyy tuotteenhallinnan jäljitettävyys. Tämä tarkoittaa sitä, että tiedetään mitkä komponenttien versiot muodostavat minkäkin ohjelmistoversion. Suurissa projekteissa versionhallinta monimutkaistuu helposti, joten voi olla aiheellista nimetä versionhallinnan vastuuhenkilö erikseen. (Haikala & Mikkonen 2011, 169 – 171; Lehtimäki 2006, 99.)

Tässä projektissa ohjelmiston kehittäminen tapahtui jatkuvan integraation hengen mukaisesti, joten versionhallinta oli äärimmäisen yksinkertaista. Työskennellessäni päätin siis aina etukäteen, minkälaisen osakokonaisuuden koodaan ja testaan kerralla. Säilytin yhden ja saman ohjelman sisällä vanhaa koodia kommenttien muodossa, mikäli uudet muutokset eivät aina toimisi odotetusti. Varmuuskopiointia ei tule unohtaa edes näin pienessä projektissa (Lehtimäki 2006, 101), joten päivitin ohjainohjelman kaksi varmuuskopiota aina työviikon päätteeksi.

3.6 Testaussuunnittelu

Projektin alkupuolella tulisi suunnitella edes jotenkuten mitä, milloin ja miten testataan mitäkin (Lehtimäki 2006, 170). Pienessä projektissa yksinkertainen testaussuunnitelma on yleensä riittävä, joten sen voi sisällyttää osaksi muuta projektia koskevaa dokumentaatiota (Haikala & Mikkonen 2011, 217). Itse sisällytin suurpiirteisen testaussuunnitelman projektisuunnitelmaan. Päätin myös käyttää yksinkertaisia välineitä ja suhtautua testaamiseen jatkuvana ja luontevana osana ohjelman tuottamista. Tämä vaikutti järkevältä tavalta välttää turhaa

monimutkaisuutta. Lisäksi kykenin täten minimoimaan riskiä siitä, että ongelmat jäävät huomaamatta ennen kuin on liian myöhäistä (Haikala & Mikkonen 2011, 198). Ohjelman toiminnallisuuden testaaminen tapahtui ihan vain ajamalla ohjelmaa ja tutkimalla, toimiiko se odotusten mukaisesti. Tätä täydensin käymällä läpi itsessään ohjelmakoodia aika ajoin, sillä sen hyvälaatuisuus oli minulle tärkeää.

Ohjelman kehittäminen tapahtui iteroivasti, minkä erinomaiset testausolosuhteet mahdollistivat. Kykenin asiakasedustajani valvonnan alaisena testaamaan ohjainohjelmaa sen todellisessa käyttöympäristössä päivittäin. Lisäsin ohjelmakoodiin runsaasti välitulostuksia komentokehoteikkunaan, mikä oli yksinkertainen mutta tehokas keino tarkastella ohjelman toimintaa ja paikallistaa virheitä. Sarjaporttikommunikoinnin toimivuuden tarkasteleminen vaati kuitenkin erillisen ohjelman käyttämistä.

Testaamisen määrä on aina kompromissi, sillä ainuttakaan ohjelmaa ei koskaan voida todeta virheettömäksi täydellä varmuudella (Haikala & Mikkonen 2011, 210). Lisäksi tässä projektissa ohjainohjelman käytettävyys oli tärkeää, joten sovimme asiakasedustajan kanssa testaamiskäytännöistä tämän huomioiden. Ohjelman testaaminen oli yksin minun vastuullani siihen saakka, kunnes kaikki ohjelman toiminnallisuus oli testieni perusteella virheetöntä. Sen jälkeen luovutin ohjelman asiakasedustajalle käyttäjätestaamista varten. Tämä tarkoittaa yleensä koeryhmän käyttämistä ohjelman käytettävyyden arvioimiseksi (Kuutti 2003, 68). Minun ei tarvinnut kerätä testaajia eikä suunnitella etukäteen testitapauksia. Projektin asiakasläheisyys mahdollisti sen, että aidot käyttäjät saivat ohjainohjelman oikeaan tuotantokäyttöön. He yksinkertaisesti tekivät normaalisti töitä ohjelmallani. Asiakasedustaja välitti heidän palautteensa minulle, minkä perusteella korjasin sekä käytettävyysongelmat että käytössä ilmenneet virheet.

4 OHJELMAN SUUNNITTELU

4.1 Teknologia

4.1.1 Ohjelmat ja tekniikat

Minulla ei ollut projektissa käytössä yhtään raharesursseja, joten valikoin toteutusteknologiaksi ilmaisohjelmia. Pyrin myös pitämään asiat yksinkertaisina käyttämällä itselleni tuttuja tekniikoita. Toteutuskieleksi valitsin ohjelmointikielistä parhaiten hallitsemani C#:n ja sen ohella .NET Frameworkin version 4.5. Versiosta 2 alkaen .NET Frameworkiin on sisällytetty valmis sarjaporttiluokka (Microsoft b). Kuvassa 1 havainnollistetaan SerialPort-oliota. Ohjainohjelmaa tullaan käyttämään Windows 7-käyttöjärjestelmällä, joten mielestäni oli järkevää hyödyntää myös .NET Frameworkiin sisältyvää Windows Forms -tekniikkaa käyttöliittymän rakentamiseen (Microsoft a).

```

//Serial port
private static SerialPort serialport; //Serialport
public static bool open = false; //Is the port open?
public static bool inProgress = false; //Is the port occupied?

//Constants
private static ushort STX = 2; //Start of text
private static ushort DLE = 16; //Data link escape
private static ushort ETX = 3; //End of text
private static ushort NAK = 21; //Negative acknowledgement

//For this class' use
public static byte controlByte; //Used to track the progress of communication and whether it is working as it should
public static byte BCC; //Blockchek; used in the protocol
public static int code; //Numeral code used for identifying different requests and messages
public static int max = 30; //Maximum length for data arrays

//Opening and closing the communication

public static void OpenCommunication() //Opening the communication
{
    serialport = new SerialPort("COM1");

    //Mikä on kieli? Muista tarkistaa!
    //Tarkista muutkin arvot!
    serialport.Encoding = Encoding.GetEncoding("ISO-8859-1"); //Needed to translate characters etc.
    serialport.BaudRate = 9600;
    serialport.ReadTimeout = 50000;
    serialport.Parity = Parity.None;
    serialport.DataBits = 8;
    serialport.StopBits = StopBits.One;
    serialport.DtrEnable = true;
    serialport.RtsEnable = true;
    serialport.Handshake = Handshake.None;

    serialport.Open();

    open = true;
}

```

Kuva 1: Esimerkki SerialPort-olion käytöstä ohjelmassa

Toteutustyökaluksi itse ohjelman kehittämiseksi valitsin Visual Studio 2015 Communityn, sillä minulla on aiempaa kokemusta ohjelmistosta. Sen tuki C#-kielelle sekä .NET Frameworkille on todella hyvä, mikä helpotti ja tehosti työskentelyäni. Lisäksi tämän ohjelmistoversion käyttäminen oli kaltaiselleni yksityishenkilölle ilmaista (Microsoft c). Sarjaporttien välillä liikkuvan datan seuraamiseen vaadittiin oma työkalunsa. Käytin HHD Softwaren sarjaportin monitorointiohjelman ilmaista kokeiluversiota, joka osoittautui käytössä oivalliseksi. Ohjelmalla oli mahdollista tulostaa näytölle sekä tietokoneen sarjaportin lähettämä että vastaanottama data.

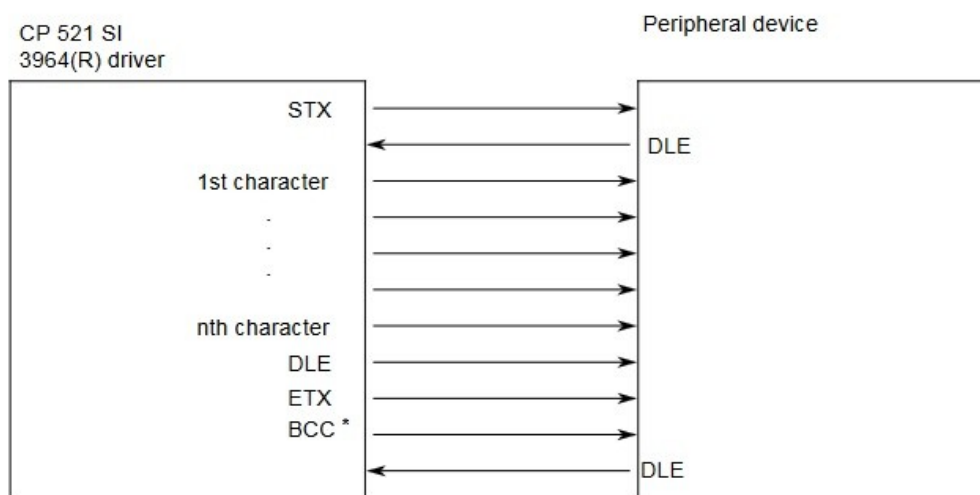
4.1.2 Siemensin 3964R-protokolla

Tietokone ja PLC tarvitsevat yhteiset säännöt, joiden perusteella ne voivat kommunikoida keskenään. Tässä projektissa käytössä oli PLC:n laitevalmistajan eli Siemensin standardoitu 3964R-protokolla. R-kirjain sen nimessä tarkoittaa, että datasiirrossa on käytössä ylimääräinen tarkistusmerkki (block check, BCC) datan virheettömyyden takaamiseksi. Protokolla soveltuu hyvin pienten datamäärien

liikuttamiseen, eikä handshake-tekniikan käyttäminen säännöstelyyn ole käytännössä koskaan tarpeellista. Molemmille sarjaportteille on määriteltävä samanlaiset parametrit lukuun ottamatta sitä, että toisella on oltava korkeampi prioriteetti. Oleellinen tieto kuljetetaan laitteelta toiselle datapaketteina kontrollimerkkien avulla. Nämä merkit ovat STX (start of text), ETX (end of text), DLE (data link escape) ja NAK (negative acknowledgement). (Siemens 1993, 220, 224 – 225; Siemens 2001.)

Käytännössä ensimmäinen laite avaa keskusteluyhteyden lähettämällä toiselle STX-merkin. Tämä on ilmoitus siitä, että laite on aikeissa aloittaa datapaketin lähettämisen. Vastaanottava laite palauttaa DLE-merkin osoituksena siitä, että on ymmärtänyt pyynnön ja on valmis datasiirtoon. Keskusteluyhteys on täten auki ja lähettävä laite siirtää yhden merkin kerrallaan puskuristaan vastaanottavalle laitteelle, kunnes puskuri on tyhjä. Tämän jälkeen tulee vastaanottavalle laitteelle vielä lähettää kolme kontrollimerkkiä järjestyksessä, jotka ovat ETX, DLE ja BCC. Aivan lopuksi vastaanottava laite vastaa vielä yhdellä DLE-merkillä osoituksena siitä, että datapaketti on vastaanotettu onnistuneesti ja virheettömänä. (Siemens 1993, 226 – 228.) Kuvassa 2 on esitetty tämä kommunikointiprosessi yksinkertaisessa muodossa. Kuvassa 3 on esitetty datapaketin siirtämisen ohjelmallinen toteutus.

Example of error-free transmission:



* BCC only in the 3964R transmission protocol

Figure 8-7. Error-Free Data Interchange (Send)

Kuva 2: 3964R-protokollaa noudattavaa keskustelua havainnollistava kuva (Siemens 1993, 227)

Protokollasta poikkeaminen johtaa auttamatta kommunikoinnin epäonnistumiseen. Kumpi tahansa laite voi ilmoittaa toiselle jonkinlaisesta virheestä kesken keskustelun lähettämällä NAK-merkin. Tällaisessa tilanteessa keskustelu yritetään aloittaa uudelleen alusta niin monta kertaa, kuin sarjaportin parametreihin on asetettu. Mikäli jostakin syystä molemmat laitteet yrittävät avata keskusteluyhteyden samaan aikaan, myöntyy alemman prioriteetin omaava laite vastaanottamaan korkeamman prioriteetin laitteen datapaketin. (Siemens 1993, 228, 230.)

```
private static void WriteData(char[] data) //Writing datapacket
{
    BCC = 0;
    char[] buffer = new char[1];

    for (int i = 0; i < data.Length; i++)
    {
        buffer[0] = data[i];
        serialport.Write(buffer, 0, buffer.Length);
        BCC = (byte)(BCC ^ data[i]);

        //16 is to be sent twice or logic will understand it as mere DLE
        if (data[i] == (char)16)
        {
            serialport.Write(buffer, 0, buffer.Length);
            BCC = (byte)(BCC ^ (ushort)data[i]);
        }
    }

    buffer[0] = (char)DLE;
    serialport.Write(buffer, 0, buffer.Length);
    BCC = (byte)(BCC ^ DLE);

    buffer[0] = (char)ETX;
    serialport.Write(buffer, 0, buffer.Length);
    BCC = (byte)(BCC ^ ETX);

    buffer[0] = (char)BCC;
    serialport.Write(buffer, 0, buffer.Length);

    controlByte = 11;
}
```

Kuva 3: Metodi datapaketin lähettämiseksi PLC:lle osana keskustelun toteutusta

4.2 Datapaketeista

Datapakettien kokoaminen ja käsitteleminen eivät vaatineet mitään erillisiä tekniikoita sinällään, mutta oli ongelmallinen kokonaisuus. Tietokoneen ja PLC:n välillä liikkuu monipuolista tietoa jalaskoneen toiminnan ohjaamiseksi ja seuraamiseksi aina byte eli tavu kerrallaan. Protokollasta poiketen suurin osa siirrettävästä datasta on todellisuudessa 16 bitin eli kahden tavun kokoisia lukuja. Edellisessä ohjainohjelmassa merkit siirrettiin char-tietotyyppisinä, joten päätin tässä ohjainohjelmassa toimia samoin välttääkseni mahdollisia odottamattomia ongelmia. (Jalasta 2005.) Datapakettien muoto ja sisältö on ohjelmoitu tarkkaan PLC:n sisälle, mutta tarvittavan tiedon kerääminen edellisestä ohjainohjelmasta (Jalasta 2005) oli hidasta ja raskasta. Lisäksi minun oli tutustuttava tietokonemerkistöihin tarkemmin.

ASCII-merkistö kehitettiin englannin kielen ilmaisemiseen, johon 128 merkkipaikkaa alun perin riittivät. Lisäksi merkkipaikoille 0:sta 32:een sekä 128 sijoitettiin laitteiden komentamiseen käytetyt kontrollimerkit ja mm. välilyönnin ja tabulaattorin painallusten esittäminen. Merkkien ilmaisemiseen riittivät byten kahdeksasta bitistä seitsemän. Tästä syystä merkistössä oli todellisuudessa tilaa 255 erilaiselle merkillle. Eri variaatiot ASCII-merkistöstä ovat syntyneet yksinkertaisesti siitä syystä, että eri tahot täyttivät nämä vapaat merkkipaikat omiin tarpeisiinsa sopivilla merkeillä. (Spolsky 2003.) Tietokoneen ja PLC:n tuli käyttää samaa merkistöä, sillä muuten ne tulkitsivat ihmissilmälle samoja merkkejä eri tavoin. Unohdin alun perin varmistaa tämän, mutta datapakettien kääntäjän testaamisessa ilmenneet epämääräiset virheet muistuttivat asiasta. Onneksi SerialPort-oliolle oli mahdollista asettaa encoding eli merkistö, jota sen tulee käyttää (Microsoft b). Ensin minun oli kuitenkin kulutettava hieman aikaa selvittääkseni tietokoneen käyttämä merkistö vertaamalla testituloksia eri ASCII-taulukoihin.

Datapaketit ovat aina samankokoisia eli ne sisältävät 30 kappaletta byten pituisia merkkejä. Jokaisen datapaketin ensimmäisenä lukuna on yksi muutamista vaihtoehtoisista viestinumeroista. Niiden perusteella PLC ymmärtää, millaista tietoa datapaketti edustaa ja mitä sillä tulee tehdä. Toisena merkinä datapaketissa on aina nolla, minkä lisäksi jokaisen datapaketin kaksi viimeistä merkkiä ovat yhden 16 bitin pituisen BCC-kontrollimerkin kaksi puolikasta. Vastaanotettuaan datapaketin tulee

laitteen aina laskea BCC määritysten mukaisesti ja verrata sitä saamassaan datapakettissa olevaan BCC:hen. Saapuneen datan todetaan olevan virheetöntä, mikäli BCC:t täsmäävät. (Alber sähköposti 20.6.2016; Jalasta 2005)

Sain jalaskoneen PLC:n ohjelmoijalta sähköpostitse (Alber 20.6.2016) listan sen käyttämistä viestinumeroista:

- 10 eli tilaustiedot logiikalle; ohjainohjelma lähettää uuden jalastilauksen PLC:lle
- 11 eli tilaustiedot logiikalta; ohjainohjelma pyytää PLC:ltä tiedot tämänhetkisestä jalastilauksesta
- 12 eli tiedonkeruu; ohjainohjelma pyytää PLC:ltä tiedot sen tämänhetkisestä tilasta
- 13 eli parametrit logiikalle; ohjainohjelma lähettää uudet asetustiedot PLC:lle
- 14 eli parametrit logiikalta; ohjainohjelma pyytää PLC:ltä tämänhetkiset asetustiedot
- 15 eli nollaus; ohjainohjelma lähettää PLC:lle käskyn nollata tämänhetkisen toimintansa
- 16 eli I/O-tiedot logiikalta; ohjainohjelma pyytää PLC:ltä sen tämänhetkiset I/O-tilatiedot
- 17 eli käsiajo; ohjainohjelma lähettää PLC:lle ilmoituksen jonkin käsiajon käytöstä
- 18 eli korjaus; ohjainohjelma lähettää PLC:lle tiedot, joiden perusteella sen tulee korjata kuljettimien ja pistoolien asentoa välittömästi

Suurin ongelma oli se, että itse datapakettien sisällöstä ei ollut yhtä helppo saada selvyyttä. Edellisen ohjainohjelman koodia tarkkaan tutkimalla löysin kuitenkin suurimman osan tarvitsemistani tiedoista (Jalasta 2005). Loput oli pakko selvittää kokeilemalla erilaisia arvoja datapakettien tuntemattomilla merkkipaikoilla ja tutkimalla, miten jalaskone reagoi niihin. Päättely, systemaattinen kokeilu sekä asiakasedustajan jalaskoneen tuntemus auttoivat minua muodostamaan paperille kunnolliset taulukot jokaisen erilaisen datapaketin sisällöstä. Tämä oli tärkeää erityisesti siitä syystä, että erilaisissa datapaketeissa 16-bittiset luvut on sijoitettu eri kohtaan (Jalasta 2005). Ohjelman on osattava erikseen halkaista juuri oikeissa kohdissa olevat luvut kahteen byteen datapaketin viestinumeron perusteella (Kuva 4). Tietysti arvojen sijaitseminen väärissä kohdissa datapakettia johtaa yksinkertaisesti jalaskoneen virheelliseen toimintaan.

```

char[] re;
List<char> l = new List<char>();
byte upper;
byte lower;

for (int i = 0; i < u.Length; i++)
{
    upper = (byte)(u[i] >> 8);
    lower = (byte)(u[i] & 0xff);

    if ((code == 17) || (code == 10))
    {
        if ((i == 0) || (i == 1))
        {
            l.Add(Convert.ToChar(lower));
        }
        else
        {
            l.Add(Convert.ToChar(upper));
            l.Add(Convert.ToChar(lower));
        }
    }
    else if (code == 13)
    {
        if ((i == 0) || (i == 1) || (i == 2) || (i == 3) || (i == 10) || (i == 11))
        {
            l.Add(Convert.ToChar(lower));
        }
        else
        {
            l.Add(Convert.ToChar(upper));
            l.Add(Convert.ToChar(lower));
        }
    }
    else
    {
        //...
    }
}

```

Kuva 4: Katkelma datapakettien sisällön 16-bittisistä luvuista merkeiksi (char) kääntävästä koodista

Alla listattuna muistiinpanot eräästä datapaketista, joka sisältää PLC:lle lähetettävät tilaustiedot:

- [0] = 10
- [1] = 0
- [2] = HighByte(Lavan kappalemäärä) = 1000
- [3] = LowByte(Lavan kappalemäärä)
- [4] = HighByte(Jalakset per lava) = 3
- [5] = LowByte(Jalakset per lava)
- [6] = HighByte(Ylälaudan pituus) = 1200
- [7] = LowByte(Ylälaudan pituus)
- [8] = HighByte(Ylälaudan leveys) = 100
- [9] = LowByte(Ylälaudan leveys)
- [10] = HighByte(Kuljettimen leveys säätö ohje) = YlälaudanPituus / 2 * 2 = 1200

- [11] = LowByte(Kuljettimien leveys säätö ohje)
- [12] = HighByte(Pistoolien leveys säätö ohje) = $\text{YlalaudanPituus} / 2 * 2 = 1200$
- [13] = LowByte(Pistoolien leveys säätö ohje)
- [14] = HighByte(Klopit per jalas) = 3
- [15] = LowByte(Klopit per jalas)
- [16] = HighByte(5) (Mekaanikkaa koskeva vakio)
- [17] = LowByte(5)
- [18] = HighByte($\text{Ylalaudanleveys} \text{ div } 2$) = $100 / 2 = 50$
- [19] = LowByte($\text{Ylalaudanleveys} \text{ div } 2$)
- [20] = HighByte(200) (Mekaniikkaa koskeva vakio)
- [21] = LowByte(200)
- [22] = HighByte(Naulauskuva) = 63 tai 51
- [23] = LowByte(Naulauskuva)

Listaan on merkitty jokaisen arvon paikka taulukossa sekä tieto, jota kukin merkki edustaa. Lisäksi siihen on sisällytetty esimerkkiarvoja, joita asiakasedustaja minulle antoi testipaketin muodostamista varten. HighByte ja LowByte ovat termejä edellisen ohjainohjelman koodista (Jalasta 2005), jotka merkitsevät yhden 16 bitin pituisen luvun alku- ja loppupuolikkaita. Paikoille 18 ja 19 sijoittuvan arvon relevanssia en vieläkään tiedä, mutta ylalaudan pituuden jakaminen kahdella on kaikissa testeissä johtanut odotustenmukaiseen jalaskoneen toimintaan. Kuljettimien ja pistoolien leveydensäätöohjeiden jakaminen ja sitten kertominen kahdella puolestaan on turha laskutoimitus, joten jätin sen itse toteuttamatta.

4.3 Arkkitehtuurisuunnittelu

Ohjelmiston suunnitteluvaiheen tavoitteena on pähkinänkuoressa tarkoittaa määrittelyvaiheen tuloksia siihen pisteeseen, että varsinainen toteutus voidaan aloittaa (Suomen Automaatioseura ry 2005, 48). Omassa projektissani tämä tapahtui melko käyttäjäkeskeisesti (Kuutti 2003, 149), sillä toimin koko ajan tiiviissä yhteistyössä asiakkaan kanssa. Suunnittelu jaetaan yleensä erilaisiin osiin, jotta kokonaisuuden hallitseminen olisi helpompaa ja tehokkaampaa. Muutenkin turhan monimutkaisuuden välttämistä on vain hyötyä. (Haikala & Mikkonen 2011, 181 – 182; Pohjonen 2002, 32.) Itse jaoin suunnittelun raa'asti arkkitehtuurisuunnitteluun ja luokkasuunnitteluun. Pyrin pitämään ohjainohjelman mahdollisimman yksinkertaisena.

Arkkitehtuurin voidaan ajatella tarkoittavan ohjelmiston kiinteää runkoa. Arkkitehtuurisuunnitelma on koko toteutuksen pohjapiirros, joka kuvaa rakennetta. Lisäksi se on tärkeä väline yhteisymmärryksen luomiseksi eri sidosryhmien välille, joten sen tulisi olla selkeä ja kattava. (Haikala & Mikkonen 2011, 178 - 179; Koskimies & Mikkonen 2005, 41.) Arkkitehtuurien kuvaamiseen on olemassa erikseen mallinnusmenetelmiä (Suomen Automaatioseura ry 2005, 48), mutta koin sellaisten tutkimisen ja käyttämisen turhaksi. Edes asiakasedustajani ei tarvinnut tietoa arkkitehtuurista, mutta informoin häntä siitä yleisellä tasolla joka tapauksessa.

Arkkitehtuurisuunnittelun hyödyistä merkittävin on sen mahdollistama monimutkaisuuden hallitseminen. Arkkitehtuuritasolla pystytään jakamaan ohjelma osiin, joiden kehittämiseen voidaan keskittyä erikseen. Näiden komponenttien välisten suhteiden ja riippuvuuksien tunnistaminen, määrittelemine ja hallitseminen ovat arkkitehtuurisuunnittelussa tärkeä kokonaisuus. Yleisesti ottaen pyritään siihen, että komponenttien välillä olisi mahdollisimman vähän riippuvuuksia. Se selkeyttää koko rakennetta sekä mahdollistaa helpomman muunneltavuuden. Arkkitehtuuritasolla voidaan lisäksi kokeilla ja vertailla erilaisia rakenneratkaisuja kätevästi. (Koskimies & Mikkonen 2005, 16 – 19, 27, 75, 95.)

Ainakaan tässä projektissa arkkitehtuurisuunnittelu ei vaikuttanut missään vaiheessa niin monimutkaiselta kuin kirjallisuus antaa ymmärtää. Minulle suunnittelun pääasiat olivat yksinkertaisuus sekä vaatimusmäärittelyyn kirjattu tarve siitä, että sarjaportit korvataan tulevaisuudessa toisenlaisella toteutuksella. Pyrin huomioimaan tämän siten, että sarjaporttikommunikoinnin toteuttava osa olisi mahdollisimman riippumaton muusta ohjelmasta. Arkkitehtuurisuunnitelmani perustui Visual Studion valmiiksi uuden Windows Forms -projektin luomisen yhteydessä generoimaan pohjarakenteeseen. Tämä ns. valmispohja oli sekä helpoin että luontevin keino toteuttaa käyttöliittymä. Jaoin ohjainohjelmani muuten karkeasti muutamaan komponenttiin, jotka olivat käytännössä luokkia.

En nähnyt mitään varsinaista syytä erottaa ohjelman käyttöliittymää ja toiminnallisuutta toisistaan, joten päätin toteuttaa ne toisiinsa sidoksissa Windows Forms -valmisluokkien avulla. Niiden ohella ohjelmassa on tietysti Program-luokka,

jonka ainoa tarkoitus on käynnistettäessä luoda ohjelma. Sarjaporttikommunikoinnin toteutin kokonaan omana luokkana, johon muu ohjelma on kosketuksissa vain kysyessään sarjaportin tämänhetkistä tilaa tai kutsuessaan metodia datapaketin lähettämiseksi. Tällöin koko luokka on helppo korvata uudella tulevaisuudessa, eikä rajapinta tarvitse välttämättä lainkaan muutoksia. Näiden komponenttien lisäksi totesin tarvitsevani itsenäisen ns. tietovaraston, jonne kaikki muut komponentit voivat tallettaa tärkeää tietoa tai hakea sitä. Suunnittelin luokkien sisäisen toiminnan melko suurpiirteisesti, joten esittelen ne suoraan toteutuksen näkökulmasta.

5 TOTEUTUS JA TOIMINNALLISUUS

5.1 Luokat

5.1.1 Form1.Designer

Kyseessä on toinen kahdesta valmisluokasta, jotka Visual Studio loi automaattisesti uuden Windows Forms -projektin luomisen yhteydessä. Windows Forms -tekniikassa form on käyttöliittymänäkymän tyhjä pohja, johon lisätään datasyötteitä vastaanottavia sekä dataa esittäviä visuaalisia elementtejä (Microsoft d). Form1.Designer-luokka sisältää sekä Form1:n että kaikkien sen elementtien luomisen ja niiden parametrien alustamisen. Koko luokka muodostaa käytännössä yhden valtavan metodin, jota Form1-luokka kutsuu, kun koko ohjelma käynnistetään ja käyttöliittymä ns. maalataan näkyviin ensimmäisen kerran.

Ohjelmassa on runsaasti elementtejä, joiden tehtävänä on näyttää käyttäjälle PLC:ltä vastaanotettua dataa, mutten halunnut luoda turhaa riippuvuutta Form1.Designer-luokan ja SP-luokan välille. Tilaus- ja asetustiedot voidaan hakea ja sijoittaa Data-luokasta, mutta PLC:n tilatietoja kuvaaviin elementteihin kirjataan aluksi ”0”. Niihin oikean sisällön hakeminen PLC:ltä tapahtuu Form1-luokassa.

5.1.2 Form1

Kyseessä on toinen kahdesta valmisluokasta, jotka Visual Studio loi automaattisesti uuden Windows Forms -projektin luomisen yhteydessä. Windows Forms -tekniikassa form ja elementit reagoivat käyttäjän toimiin synnyttämällä tapahtumia (Microsoft d). Kaikkien näiden tapahtumien toteutukset sijoittuvat Form1-luokkaan automaattisesti. Päätin siksi käyttää tätä luokkaa koko ohjelman toiminnallisuuden toteuttamiseen.

Luokka sisältää tapahtumien lisäksi runsaasti erilaisia metodeja, jotka olen ryhmitellyt huolellisesti ohjelmakoodiin eri kokonaisuuksiksi kommenttien avulla:

- Form1:n konstruktori
- Tietoa esittävien elementtien muokkaustilojen avaaminen ja sulkeminen käyttäjälle eri näkymissä
- Tietoihin tehtyjen muutosten hyväksyminen ja kumoaminen
- SP-luokan rajapintaa käyttävät metodit, jotka joko keräävät tarvittavat tiedot oikeanlaisiksi datapaketeiksi lähetystä varten (Kuva 6) tai pyytävät datapaketin ja käsittelevät sen
- Ajastin, joka sekä päivittää kellon käyttöliittymässä että hakee tämänhetkiset PLC:n tilatiedot SP-luokalta joka sekunti
- Sekalaiset ns. helper-metodit, joiden avulla muut metodit toimivat luokassa sujuvammin ja ovat sisäiseltä rakenteeltaan selkeämpiä
- Valikkonappien tapahtumat ja toiminnallisuus
- Käyttöliittymän eri näkymien syötelaatikoiden tai valintanappien tapahtumat ja toiminnallisuus

```
private void DataSendTilaus() //Create array of Tilaus values and send them to the PLC
{
    SP.inProgress = true;
    ushort[] receive;
    List<ushort> l = new List<ushort>();

    l.Add(Convert.ToInt16(10)); //Code
    l.Add(Convert.ToInt16(0));
    l.Add(Convert.ToInt16(txbTilaus1.Text)); //Lavan kappalemäärä
    l.Add(Convert.ToInt16(txbTilaus2.Text)); //Jalakset per lava
    l.Add(Convert.ToInt16(txbTilaus3.Text)); //Ylälaudan pituus
    l.Add(Convert.ToInt16(txbTilaus4.Text)); //Ylälaudan leveys
    l.Add(Convert.ToInt16(txbTilaus3.Text)); //Kuljettimen leveys säätö ohje ((Ylälaudan pituus / 2) * 2 = Ylälaudan pituus)
    l.Add(Convert.ToInt16(txbTilaus3.Text)); //Pistoolien leveys säätö ohje (same as above)
    l.Add(Convert.ToInt16(txbTilaus5.Text)); //Klopit per jalas
    l.Add(Convert.ToInt16(5));
    int temp = Convert.ToInt16(txbTilaus4.Text) / 2; //Ylälaudan leveys / 2
    l.Add(Convert.ToInt16(temp));
    l.Add(Convert.ToInt16(200));
    //Naulauskuva is either XX00XX (51) or XXXXXX (63) depending on Klopit per jalas
    if (Convert.ToInt16(txbTilaus5.Text) == 2)
    {
        l.Add(51);
    }
    else
    {
        l.Add(63);
    }

    receive = SP.DataSendReceiveA(10, l);
}
```

Kuva 5: Esimerkki datapaketin kokoamisesta tilaustietojen lähettämiseksi PLC:lle Form1-luokassa

5.1.3 Data

Data-luokka sisältää kokoelman sellaista tietoa ja arvoja, jotka ovat tärkeitä ohjainohjelman sisäisen toteutuksen sekä PLC:n toiminnan kannalta. Osa arvoista on kovakoodattu luokkaan, loput se hakee PLC:ltä ohjelman käynnistämisen yhteydessä. Tämä tapahtuu Form1-luokan konstruktoria jo ennen käyttöliittymän

luomista, joten Form1.Designer-luokka voi hakea tietoa sisällöksi joihinkin elementteihin välittömästi.

Ohjelman käytön aikana Data-luokassa olevan tiedon tärkein käyttötarkoitus on datapakettien kokoamisessa. Luokka toteutetaan singletonina eli siitä on olemassa vain yksi ilmentymä koko ohjelmassa. Näin eri luokat ovat kosketuksissa varmasti samaan tietoon. Kuvassa 7 havainnollistetaan Data-luokan ilmentymän luomista. Metodissa nähdään myös, miten luokka käyttää SP-luokan rajapintaa hakeakseen tarvitsemansa tiedot PLC:ltä. Tämä luo riippuvuuden luokkien välille, mutta Data-luokan hyödyt olivat mielestäni riippuvuudesta mahdollisesti seuraavia haittoja suuremmat.

```
public static Data CreateData()
{
    Data d = new Data();
    SP.OpenCommunication();

    //Upon the start of the program we create this class to store all the needed data locally
    //The data is always stored inside the PLC

    Console.WriteLine("Asetukset!");
    List<ushort> l = new List<ushort>();
    l.Add(14); //Asetukset from logic
    l.Add(0);
    ushort[] receive;

    receive = SP.DataSendReceiveA(14, 1);

    d.servonEteenNopeus = receive[2];
    d.servonTaakseNopeus = receive[3];
    d.servonOffset = receive[4];
    d.kuljettimienOffset = receive[5];
    d.pistoolienOffset = receive[6];
    d.kehtoYlhaalla = receive[7];
    d.kehtoAlhaalla = receive[8];
    d.ketjunVenymisenKorjaus = receive[9];

    Console.WriteLine("Tilaustiedot!");
    l = new List<ushort>();
    l.Add(11); //Tilaustiedot from logic
    l.Add(0);

    receive = SP.DataSendReceiveA(11, 1);

    d.lavojenKappalemaara = receive[2];
    d.jalaksetPerLava = receive[3];
    d.ylalaudanPituus = receive[4];
    d.ylalaudanLeveys = receive[5];
    d.klopitPerJalas = receive[8];

    return d;
}
```

Kuva 6: Data-luokan alustaminen ja luominen

5.1.4 SP

SP-luokkaan kuuluu kaikki mahdollinen koko ohjelmassa, jota sarjaporttikommunikoinnin toteuttaminen vaatii. Ajatuksena on, että muut luokat ovat kosketuksissa koko luokkaan vain ns. rajapinnan kautta. Käytännössä tämä tarkoittaa sitä, että muut luokat kutsuvat `DataSendReceiveA`-metodia silloin, kun haluavat lähettää tai vastaanottaa dataa sarjaportilta. Muiden luokkien ei tarvitse kuin koota lista lukuja datapaketiksi, jotka SP-luokka kääntää oikeaan muotoon. Lisäksi muut luokat voivat pyytää sarjaportin tilan tältä luokalta. Tätä tietoa tarvitaan lähinnä, jottei ohjelma yrittäisi aloittaa uutta datasiirtoa PLC:n kanssa silloin, kun edellinen on vielä kesken.

SP-luokan sisältämiä metodeja ovat sarjaportin avaaminen ja sulkeminen, 3694R-protokollan mukaisen kommunikoinnin ohjelmallinen toteutus sekä datapakettien kääntäminen haluttuun muotoon. Pakettien lähettäminen ja vastaanottaminen on teoriassa yksinkertaista, mutta käytännössä sen toteuttaminen oli pikkutarkkaa työtä. Koodiin lipsahti helposti pieniä virheitä, joten pilkoin kommunikointiprosessin useisiin metodeihin hallitakseni monimutkaisuutta. Yhtä näistä metodeista havainnollistetaan esimerkkinä Kuvassa 8. Siinä tietokone vastaanottaa PLC:ltä dataa, vaikkakin kutsuu toista metodia (`ReadData`) vastaanottaakseen itse datapaketin. Lopuksi metodi palauttaa vastaanotetun datapaketin käännettynä koko kommunikoinnin toteuttavalle runkometodille (`DataSendReceiveA`).

```

public static ushort[] ReceiveFromLogic()    //Receiving data from logic
{
    char[] receive = new char[1];
    char[] buffer = new char[1];
    char[] temp;
    ushort[] data = new ushort[max];

    controlByte = 12;

    do
    {
        //Waiting for STX:ää
        if (serialport.BytesToRead > 0)
        {
            receive = ConstantSendReceive(STX);

            //Sending DLE
            if (controlByte == 13)
            {
                buffer[0] = (char)DLE;
                serialport.Write(buffer, 0, buffer.Length);
                controlByte = 14;
            }
            //Waiting for data
            temp = ReadData();

            //Sending DLE
            if (controlByte == 16)
            {
                buffer[0] = (char)DLE;
                serialport.Write(buffer, 0, buffer.Length);
                controlByte = 17;
            }

            //Sending NAK should there be need
            if (controlByte > 100)
            {
                buffer[0] = (char)NAK;
                serialport.Write(buffer, 0, buffer.Length);
            }

            //Translate data from characters to numbers
            data = TranslateData(temp);
            break;
        }
    } while ((controlByte != 17) && (controlByte < 100));

    return data;
}

```

Kuva 7: Esimerkki yhdestä sarjaporttikommunikoinnin toteutuksessa käytetyistä metodeista SP-luokassa

5.2 Käyttöliittymä

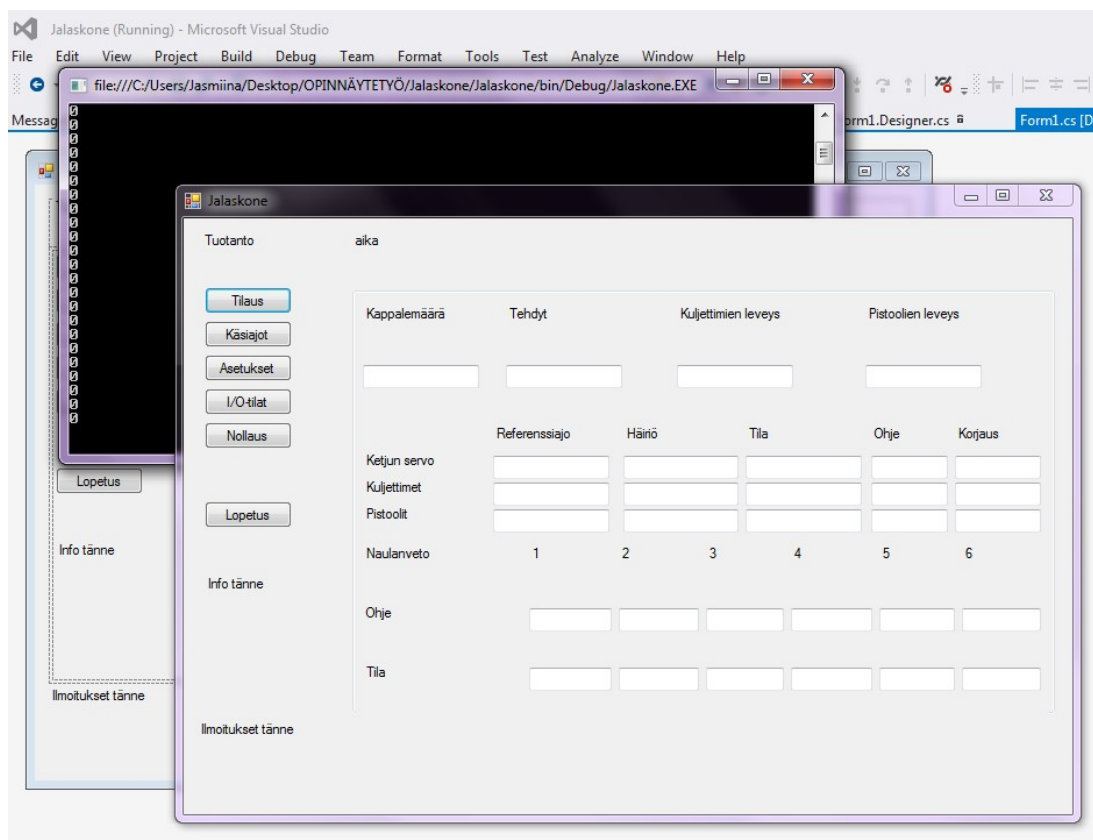
Käyttöliittymäsuunnitelman tulisi perustua kunnolliselle ohjelmarungolle ja seurata suunnitelmallisesti yhtä punaista lankaa. On tärkeää sisäistää, mitä käyttäjän on ohjelmalla tarkoitus tehdä. (Kuutti 2003, 90 - 91; Tidwell 2010, 2.) Tämä lienee helpommin sanottu kuin tehty. Omassa projektissani käyttötapaukset olivat onneksi selkeitä, minkä lisäksi sain asiakasedustajalta paljon hyödyllistä palautetta käyttöliittymäratkaisusta suunnittelun aikana. Sinällänsä minulle annettiin kuitenkin vapaat kädet. Sivuutin yleispätevät ohjeet täysin ja kyhäsin käyttöliittymästä alustavan suunnitelman ennen kuin olin päässyt ohjelmoinnissa puusta pitkään. Tähän vaikutti arkkitehtuurisuunnitelmassa tekemäni päätös, että kehitän käyttöliittymää ja toiminnallisuutta käsi kädessä. Kaikkien elementtien piti olla luotuja, että niille voi kirjoittaa minkäänlaista käyttäytymistä.

Vaativuusmäärittelyvaiheessa todettiin, että ohjainohjelman tulee ehdottomasti olla käyttäjäystävällisempi ja selkeämpi kuin edellisen. Toisaalta, yleensä käyttäjät nojaavat mieluummin valmiiseen tietotasoonsa ja kokemuksiinsa kuin opettelevat uutta (Tidwell 2010, 7). Järjestelin elementit mahdollisimman pitkälti samoin kuin edellisessä ohjainohjelmassa, jotta se olisi käyttäjille helposti lähestyttävä ja opittava (Tidwell 2010, 7, 11). Jalaskoneeseen on kytketty kosketusnäyttöinen tietokone sekä näppäimistö, joten käyttö tapahtuu joko näppäimistöllä tai liikuttamalla kursoria sormella. Painotin suunnittelussa näppäimistöä, sillä kosketusnäyttö ei toiminut kauhean hyvin.

Edellisessä ohjainohjelmassa siirtyminen eri näkymien välillä tapahtui ikkunan yläreunassa olevien painikkeiden avulla. Näkymistä puuttui koheesiota, mikä vaikeutti ohjelman sujuvaa käyttöä. Ohjelma käytti eksoottisia näppäimiä mm. tiedonlähettämisessä PLC:lle, mutta käyttöliittymässä itsessään tätä ei kerrottu mitenkään. Lisäksi ohjainohjelma ei kyennyt informoimaan käyttäjiä kunnolla. Testeissäni kävi ilmi, että se kyllä vastaanotti esimerkiksi hälytystiedot PLC:ltä, muttei silti kertonut niistä käyttäjille mitenkään. (Jalasta 2005; Tidwell 2010, 15.) Parantaakseni käytettävyyttä suunnittelin uudenlaisen navigointijärjestelmän eri näkymien vaihtelemiseksi. Lisäsin sekä käyttäjien ohjeistusta että ohjelman käyttäjälle esittämän tiedon määrää. Koheesiota ylläpidin suunnittelemalla kaikki

näkymät sekä ulkonäöltään että käyttämiltään näppäimiltä mahdollisimman samankaltaisiksi.

Alustava versio käyttöliittymän sommittelusta sisälsi jo melkein kaikki elementit, mitä lopullisessakin versiossa on. Siitä voi nähdä (Kuva 9), miten jaoin käyttöliittymän kahteen osaan. Toinen näistä on muuttumaton runko, joka ikään kuin kehystää muuttuvaa näkymää. Runkoon kuuluvat elementit ovat tällä hetkellä aktiivisen näkymän nimi ja kello (yläreuna), valikkonapit ja käyttäjää ohjeistava teksti (vasen reuna), sekä käyttäjää informoiva teksti (alareuna). Luontevan näppäimen puuttuessa lisäsin alareunaan vielä erillisen napin tietojen lähettämistä varten.



Kuva 8: Käyttöliittymän ns. pohjapiirros

Valikkonappien alla oleva infoteksti vaihtuu sen mukaan, mikä elementti on tällä hetkellä aktiivisena. Jokaisen valikkonapin kohdalla se kertoo, mikä kyseisen napin toiminto on. Kun taas on painettu jotakin napeista ja siirrytty eri näkymään kertoo infoteksti, miten ja millä napeilla toimia kyseisessä näkymässä. Käyttöliittymän

alareunassa oleva infoteksti on värillisessä laatikossa ja kertoo käyttäjälle sekä ohjelman itsensä että PLC:n tiloista jatkuvasti. Oleellista on, että tekstin ohella infotekstin taustalle sijoitettu värilaatikko vaihtaa värejä. Näin käyttäjä tietää minkälaisesta infosta on kyse jo ennen tekstin lukemista. Vihreä väri kertoo onnistuneesta ja virheettömästä toiminnasta, esimerkiksi datapaketin pääsemisestä PLC:lle onnistuneesti. Sininen väri kertoo perutusta tai ei-hälyttävästi epäonnistuneesta toiminnasta, esimerkiksi poistuttaessa jostakin näkymästä siihen tehtyjä muutoksia vahvistamatta. Punainen väri kertoo hälyttävästi epäonnistuneesta toiminnasta, esimerkiksi virheistä ja vikatiloista. Oranssi väri kertoo PLC:n omista hälytyksistä, jotka kertovat jalaskoneen mekaanisista ongelmista.

Näkymät sisältävät tekstilaatikoita, jotka sekä esittävät tietoa käyttäjälle että vastaanottavat tietoa lähetettäväksi PLC:lle. Ohjainohjelman käyttöliittymä ja toiminnallisuus ovat kuitenkin niin sitoutuneita toisiinsa, että käyn eri näkymät ja niiden toiminnallisuuden läpi erikseen. Vaihtuvia näkymiä ovat Tuotanto- eli perusnäkö, Tilausnäkö, Asetusnäkö ja Käsiäjonäkö. I/O-näkymälle ei ole toteutusta, sillä pian käyttöliittymän alustavan hahmottelun jälkeen kävi ilmi, ettei I/O-tietojen datapakettien sisällöstä ole olemassa riittävää dokumentaatiota tietojen esittämiseksi. Samasta syystä edellisessä ohjainohjelmassa ei ole ollenkaan toteutusta I/O-tietojen esittämiseksi.

5.3 Ohjelman toiminnallisuus

5.3.1 Yleistä

Perustilassa käyttäjän on mahdollista navigoida vain valikkonäppäimien välillä ylös ja alas, mikä tapahtuu nuolinäppäimillä. Valinnat tapahtuvat Enterillä, mutta eri valikkonäppäimet voi valita myös niiden nimiin merkityillä pikanäppäimillä. Korjausnäppäin siirtää käyttäjän syöttämään korjausarvoja perustilassa. Tilaus-, Asetus- ja Käsiäjonäppäimet vaihtavat perusnäkö toiseen ja avaavat kyseisen näkö arvojen syöttämisen ja muokkaamisen käyttäjälle. Nollausnäppäin lähettää PLC:lle pyynnön nollata kaikki nollattavissa olevat arvot. I/O-tilat -näppäin on toistaiseksi käyttämätön. Asiakasedustaja pyysi kuitenkin jättämään sen paikoilleen, mikäli

toiminnallisuuden toteuttaminen mahdollistuu tulevaisuudessa. Lopetusnappi pyytää käyttäjältä vahvistuksen, minkä jälkeen se sammuttaa sekä ohjelman että tietokoneen.

Eri näkymissä navigointi syötelaatikoiden tai valintanappien välillä tapahtuu nuolinäppäimillä, mutta käyttäjien pyynnöstä voidaan siirtyä laatikosta seuraavaan suoraan myös painamalla Enteriä. Kaikista muokkaustiloista (Korjaus, Tilaus, Asetukset, Käsiajot) voidaan poistua painamalla vasenta nuolinäppäintä tai Escapea. Tällöin syötelaatikoihin tehdyt muutokset myös perutaan. Muutosten vahvistaminen tapahtuu navigoimalla Lähetysnappiin ja valitsemalla se Enterillä. Tällöin käyttäjä myös palautetaan perusnäkymään. Lähetysnappi on sijoitettu navigointijärjestyksessä aina viimeisestä syötelaatikosta seuraavaksi. Syötelaatikoihin ei voi syöttää muita kuin sallittuja merkkejä (lähes poikkeuksetta numeroita). Lisäksi jokaiselle syötelaatikolle on asetettu sisäiset arvojen minimi- ja maksimirajat asiakasedustajalta kerätyn listan mukaan. Käsiajonäkymä poikkeaa hieman toteutukseltaan muista näkymistä, sillä siinä on syötelaatikoiden sijaan valintanappeja. Lisäksi napin valinta lähettää automaattisesti ilmoituksen PLC:lle, joten Lähetysnappia ei tarvita erikseen.

Käyttäjätestien aikana projektin loppuvaiheilla asiakasedustaja pyysi, että ohjainohjelma pysäyttäisi näytönsäästäjän silloin, kun ilmoituslaatikkoon tulee hälyttävä viesti (punainen tai oranssi). Tämän uuden toiminnallisuuden toteuttaminen osoittautui yllättävän ongelmalliseksi, eikä mikään ideoistani tuottanut tulosta. Keskustelupalstojen selaaminen internetissä onneksi johti minut valmiin luokan äärelle (Schulz 2007), jonka sain kätevästi liitettyä ohjelmaani uutena ScreenSaver-luokkana. Form1-luokkaan riitti pienen koodinpätkän lisääminen, joka tarkistaa ilmoituslaatikon värin joka sekunti. Mikäli se on punainen tai oranssi hälyttävän tiedon takia ja näytönsäästäjä on aktiivinen, kutsutaan ScreenSaver-luokkaa.

5.3.2 Tuotanto- eli perusnäky

Perusnäky on normaalisti näkyvissä käyttäjälle, jolloin Form1-luokka päivittää siinä näkyvän tiedon sekunnin välein. Näkymän päätarkoitus on kertoa käyttäjälle jalastuotannon etenemisestä sekä jalaskoneen tilasta. Erityisesti Naulanveto-otsikon

alla oleva ruudukko on käyttäjille tärkeä, sillä ruuduissa olevat rastit kuvaavat tällä hetkellä tehtävän jalaksen naulojen sijoittumista. Ohjeruudukossa kuvataan haluttu naulauskuvio, Tilaruudukossa kuvataan juuri tällä hetkellä käsiteltävän jalaksen naulauskuvio. Virheellisistä kuvioista hälytetään ilmoituslaatikossa (punainen väri), mutta itse ruudukosta nähdään, missä on ylimääräinen naula tai mistä se puuttuu.

Korjausnappia painamalla käyttäjä siirtyy automaattisesti Korjaus- ja +/- -otsikoiden alla oleviin neljään syötelaatikkoon. Havainnollistava kuva näkymästä tässä tilanteessa on esitetty alla kuvassa 10. Navigointi tapahtuu järjestyksessä näiden laatikoiden sekä Lähetysnapin välillä. Korjausotsikon alla oleviin laatikoihin syötetään luku 0 ja 5:n väliltä ilmaisemaan korjauksen leveyttä. Oikealla puolella oleviin pienempiin laatikoihin syötetään plus- tai miinusmerkki ilmaisemaan korjauksen suuntaa. Lähetyspainikkeella korjausarvot lähetetään PLC:lle. Korjaus tarkoittaa jalaskoneen pistoolien ja kuljettimien leveyden säätämistä nopeasti sahattavien lautojen pituuden mukaan. Tämä oli uusi vaatimusmäärittelyssä esitetty toiminnallisuus, jonka asiakasedustaja pyysi upottamaan perusnäkykseen.

Tuotanto 13:19 - - 22.9.2016

Korjaus (F1)
Tilaus (F2)
Asetukset (F3)
Käsiajot (F4)
Nollaus (F5)
I/O-tilat (F6)
Lopetus (Escape)

Aseta korjausarvot navigoiden nuolilla. Vahvista muutos (Enter) ja lähetä, tai peru muutokset (Left). Peru ja palaa perusnäkykseen myös painamalla (Escape).

	Referenssiajo	Häiriö	Tila (mm)	Ohje (mm)	Korjaus (mm)	+/-
Ketjun servo	X	0	999	0		
Kuljettimet	X	0	887	887	0	+
Pistoolit	X	0	886	887	0	+
Naulanveto						
	6	5	4	3	2	1
Ohje	X	X	X	X	X	X
Tila	X	X	X	X	X	X

Tuotantonäkymä päivitetty onnistuneesti.

Lähetä

Kuva 9: Tuotanto- eli perusnäkyvä Korjausnapin painamisen jälkeen normaalikäytössä

5.3.3 Tilausnäköymä

Tilausnappia painettaessa vaihdetaan perusnäköymä Tilausnäköymäksi (Kuva 11) ja asetetaan ensimmäinen syötelaatikko aktiiviseksi elementiksi. Näköymä näyttää käyttäjälle viimeisimmät jalasten valmistusohjeet sekä mahdollistaa uusien tietojen syöttämisen ja lähettämisen. Syötelaatikoiden välillä navigoidaan järjestyksessä, uudet tilaustiedot lähetetään PLC:lle Lähetyspainikkeella. Klopit per jalas -laatikkoon ei voida kirjoittaa arvoa, vain vaihtaa kahden vaihtoehdon välillä (2 ja 3). Naulauskuvaan käyttäjä ei voi koskea alkuunkaan, se vaihtuu automaattisesti kahden kuviovaihtoehdon välillä Klopit per jalas -laatikon arvon mukaan.

Jalaskone

Tilaus 13:19 - - - 22.9.2016

Korjaus (F1)

Tilaus (F2)

Asetukset (F3)

Käsiajot (F4)

Nollaus (F5)

I/O-tilat (F6)

Lopetus (Escape)

Aseta uudet arvot navigoiden nuolilla. Vahvista muutos (Enter) ja lähetä, tai peru muutokset (Left). Peru ja palaa perusnäköymään myös painamalla (Escape).

Lavojen kappalemäärä

Jalakset per lava

Jalaksen pituus (mm)

Jalaslaudan leveys (mm)

Klopit per jalas

Naulauskuva

	1	2	3	4	5	6
	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Tuotantonäköymä päivitetty onnistuneesti.

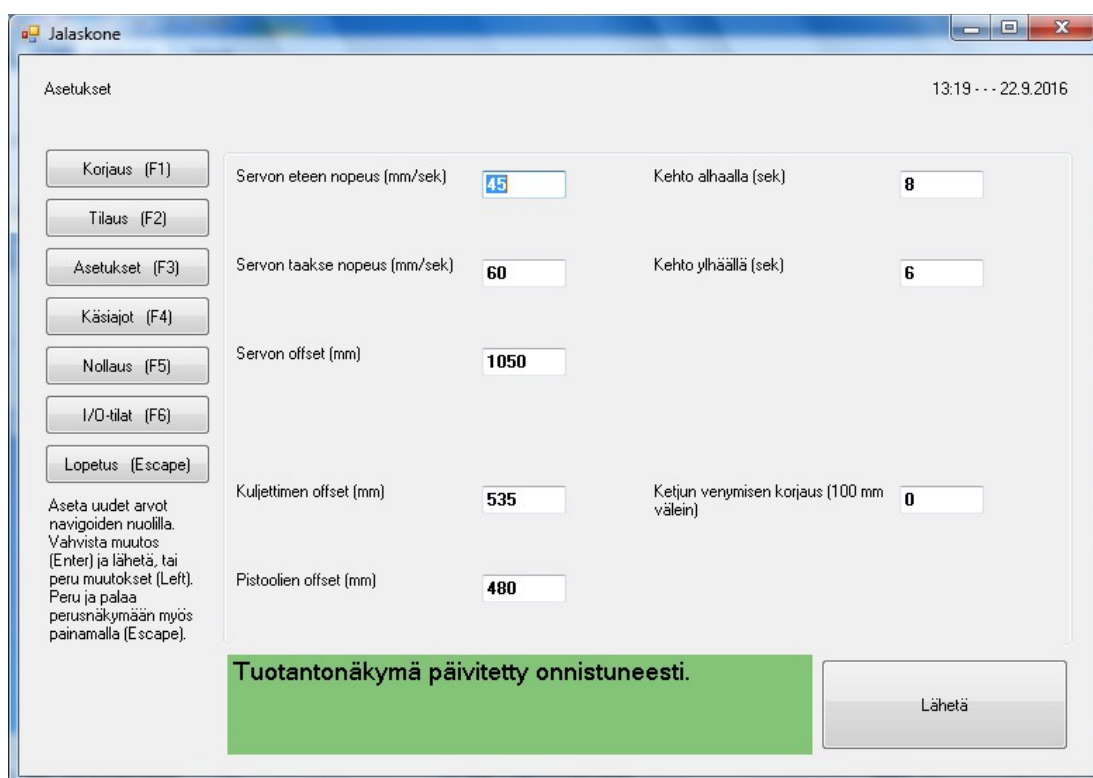
Lähetä

Kuva 10: Tilausnäköymä normaalikäytössä

5.3.4 Asetusnäköymä

Asetusnappia painettaessa vaihdetaan perusnäköymä Asetusnäköymäksi (Kuva 12) ja asetetaan ensimmäinen syötelaatikko aktiiviseksi elementiksi. Näköymä näyttää käyttäjälle jalaskoneen toimintaa ohjaavat perusasetukset ja mahdollistaa niiden

muuttamisen tarvittaessa. Syötelaatikoiden välillä navigoidaan järjestyksessä, mutta tietojen lähettäminen PLC:lle vaatii lisävahvistuksen Lähetyksenapin painamisen jälkeen. Loin ohjelmaan Form2:n, jota käytin näennäisen ponnahdusikkunan toteuttamiseen. Tässä ikkunassa käyttäjältä pyydetään salasana, jonka syöttäminen oikein vahvistaa tehdyt muutokset ja lähettää tiedot eteenpäin. Salasana on talletettu kiinteästi Data-luokkaan, sillä sen tarkoitus on vain estää muita kuin ns. sallittuja käyttäjiä tekemästä vahingossakaan muutoksia asetuksiin. Asiakasedustaja pyysi nimenomaan tällaista menettelyä, sillä siihen oli totuttu edellisessä ohjainohjelmassa.



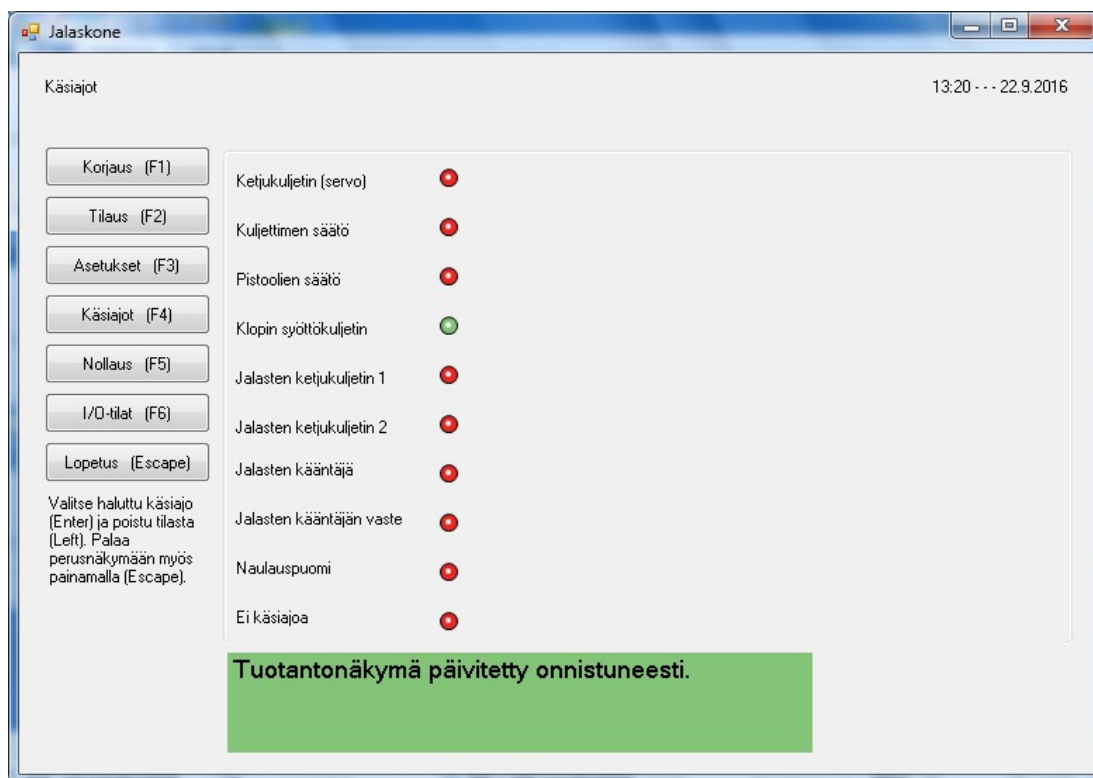
Kuva 11: Asetusnäkömää normaalikäytössä

5.3.5 Käsiäjonäkymä

Käsiäjonappia painettaessa vaihdetaan perusnäkömää Käsiäjonäkymäksi (Kuva 13) ja asetetaan ensimmäinen valintanappi aktiiviseksi elementiksi. PLC:lle lähetettävät datapaketit sisältävät numerokoodien avulla toteutetun ilmoituksen siitä, mikä käsiäjo otetaan käyttöön. Ilmoitus lähetetään aina, kun jokin vaihtoehtoista valitaan nappia painamalla. Käsiäjo tarkoittaa sitä, että käyttäjä ajaa jalaskoneen eri liikkeitä

ja toimintoja manuaalisesti. Ominaisuutta voidaan tarvita huoltotilanteissa tai ongelmanratkaisussa.

Valikkonapit toimivat kuin tavalliset radiobuttonit, mutta asiakkaan pyynnöstä periyttiin niistä uuden elementin, jolla on erilainen ulkonäkö. Napit vaihtavat väriä punaisesta vihreään sen mukaan, mikä on aktiivisena. Kun jokin vaihtoehto valitaan Enterillä, tulee napin viereen näkyville tästä teksti-ilmoitus. Valittu käsiajo voidaan vaihtaa milloin tahansa navigoimalla toisen vaihtoehdon kohdalle ja painamalla Enteriä. Näkymästä poistutaan vasemmalla nuolinäppäimellä tai Escapella kun käyttäjä on valmis käsiajojen kanssa.



Kuva 12: Käsiajonäkymä normaalikäytössä

6 YHTEENVETO

Projektin pienestä koosta huolimatta koin työskentelyn sen parissa kiinnostavaksi ja haastavaksi. Kokonaisen ohjainohjelman valmistaminen oikealle asiakkaalle oli minulle suuri vastuu. Perusteellisesta kirjallisuuden läpikäynnistä huolimatta seurasin ennen kaikkea omaa järkeäni, mutta kokemattomuuteni johti riskeihin ja ongelmiin varautumisen riittämättömyyteen. Suurilta ongelmilta vältyttiin, vaikka ajoittain saatoin jäljittää häiriön syytä usean päivän ajan. Mielestäni kuitenkin epäonnistumisesta oppii onnistumista enemmän.

Koko projektia vaikeuttivat ohjainohjelman tiukat tekniset raamit, joista ei kuitenkaan ollut saatavilla paljon tietoa. Suunnitteluvaiheen jälkeenkin aika ajoin törmäsin tilanteeseen, jossa jotakin oleellista tietoa puuttui ja oli pakko pysähtyä selvittämään asiaa. PLC:n käyttämän kommunikointiprotokollan toteuttaminen virheettömästi ja sujuvasti sekä datapakettien kokoaminen ja käsittely olivat myös minulle suuria haasteita. Käyttöliittymän kehittäminen asiakkaalle mieluisaksi pala palalta vei aikaa, mutta onnistui mielestäni silti hyvin. Aikani ohjainohjelman parissa ei ole vielä täysin päättynyt vaikka projekti onkin. Korjaan sitä vielä, mikäli virheitä tai käytettävyyso ongelmia sattuu ilmenemään. Alan myös työstää sarjaporttikommunikoinnin korvaavaa toteutusta tulevaisuudessa, kunhan sen aika joskus koittaa.

Loppujen lopuksi ylitin kuitenkin odotukseni. Vaatimusmäärittely ei pysynyt täysin muuttumattomana eikä lopputulos täysin vastannut alkuperäisiä suunnitelmia, mutta aikataulut pitivät ja hoidin mielestäni työni mallikkaasti. Ohjainohjelma voisi olla toki parempikin, mutta asiakkaani oli siihen tyytyväinen. Koin projektin hyväksi ponnahduslaudaksi sekä ohjelmistotuotantoon että sovellusalueeseen syventymiseen jatkossa.

LÄHTEET

Alber, R. Re: Kysymyksiä jalaskoneesta. Vastaanottaja: Heikkilä Jasmiina. Lähetetty 20.6.2016 klo 13.20.27. Viitattu 22.9.2016.

Haikala, I., Mikkonen, T. 2011, Ohjelmistotuotannon käytännöt. 12. uudistettu painos. Jyväskylä. Talentum media Oy..

Heikkilä, J. 2016. Tuotantopäällikkö, Porin Puulavatehdas, Versowood Oy. Pori. Henkilökohtainen haastattelu 13.5.2016. Haastattelijana Jasmiina Heikkilä. Muistiinpanot haastattelijan hallussa.

Jalasta. Ohjainohjelma. 2005. Rudy Alber.

Järvinen, J., Piispa, J. 2000. Delphi sovellusten opas. Jyväskylä. Teknolit Oy.

Koskimies, K., Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Jyväskylä. Talentum media Oy.

Kuutti, W. 2003. Käytettävyys, suunnittelu ja arviointi. Jyväskylä. Talentum media Oy.

Lehtimäki, T. 2006. Ohjelmistoprojektit käytännössä. Jyväskylä. Readme.fi.

Microsoft. 2016 a. Overview of the .NET Framework. Viitattu 20.9.2016.
<https://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.aspx>

Microsoft. 2016 b. SerialPort Class. Viitattu 19.9.2016.
[https://msdn.microsoft.com/en-us/library/system.io.ports.serialport\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.io.ports.serialport(v=vs.110).aspx)

Microsoft. 2016 c. Visual Studio Community. Viitattu 19.9.2016.
<https://www.visualstudio.com/vs/community/>

Microsoft. 2016 d. Windows Forms Overview. Viitattu 30.9.2016.
[https://msdn.microsoft.com/en-us/library/8bxxxy49h\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/8bxxxy49h(v=vs.110).aspx)

Pohjanen, R. 2002. Tietojärjestelmien kehittäminen. Jyväskylä. Docendo Finland Oy.

Schulz, K. Controlling The Screen Saver With C#. CodeProject. 8.1.2007. Viitattu: 20.9.2016. <http://www.codeproject.com/Articles/17067/Controlling-The-Screen-Saver-With-C>

Siemens. 1993. SIMATIC S5, CP 521 SI, Communications Processor Manual. 3. uudistettu painos. Viitattu 19.9.2016.
https://cache.industry.siemens.com/dl/files/578/1109578/att_21844/v1/CP521-SI_e.pdf

Siemens. 2011. What properties, advantages and special features does the procedure 3964 (R) offer? Viitattu 19.9.2016.

[https://support.industry.siemens.com/cs/document/27073039/what-properties-advantages-and-special-features-does-the-procedure-3964-\(r\)-offer-?dti=0&lc=en-WW](https://support.industry.siemens.com/cs/document/27073039/what-properties-advantages-and-special-features-does-the-procedure-3964-(r)-offer-?dti=0&lc=en-WW)

Spolsky, J. The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!). Joel on Software.

8.10.2003. Viitattu 20.9.2016. <http://www.joelonsoftware.com/articles/Unicode.html>

Suomen Automaatioseura ry. 2005. Automaatiosovellusten ohjelmistokehitys: suunnittelun työtavat, välineet ja sovellusarkkitehtuurit. Helsinki. Painomerkki Oy.

Tidwell, J. 2010. Designing Interfaces. 2. uudistettu painos. Sebastopol, Ontario. O'Reilly Media Inc.

VAATIMUSMÄÄRITTELY

Ongelma	Seuraus	Ratkaisu
Ohjainohjelma ei näytä kaikkea PLC:ltä saamaansa tietoa	Käyttäjä ei voi seurata PLC:n toimintaa kunnolla	Tunnistetaan kaikki oleellinen tieto ja esitetään se uudessa ohjainohjelmassa käyttäjälle
Ohjainohjelmalla on toimintahäiriöitä	Häiriöt haittaavat tuotantoa ja ohjelman käyttöä	Minimoidaan toimintahäiriöitä ja niistä seuraavia ongelmia uuden ohjainohjelman koodissa
Ohjainohjelma ei informoi käyttäjää toimintahäiriöistä ja ongelmista	Käyttäjä ei huomaa häiriöitä tai ongelmia ajoissa eikä voi korjata niitä sokkona	Informoidaan käyttäjää häiriöistä ja ongelmista
Ohjainohjelmassa on turhia ominaisuuksia	Käyttöliittymä ja käytettävyyks ovat monimutkaisia syytä suotta	Ei toteuteta uudessa ohjelmassa tarpeettomia toiminnallisuuksia ja ominaisuuksia
Ohjelmalla ei voi korjata jalaskoneen pistoolien ja kuljettimien asentoa tuotannon aikana	Tuotanto täytyy pysäyttää ja on käytettävä Tilausnäkyä	Lisätään toiminnallisuus, joka mahdollistaa korjausarvojen syöttämisen ja lähettämisen tuotannon aikana
Ohjainohjelmalla ei voi ottaa käyttöön käsiajoja	On käytettävä jalaskoneen mekaniikkaa, mikä on hidasta ja vaivalloista	Lisätään toiminnallisuus, joka mahdollistaa käsiajojen valitsemisen ja käyttämisen ohjainohjelman kautta
Ohjainohjelma ei vastaa käyttäjiensä käsitystä hyvästä käytettävyydestä	Ohjelman käyttö on hankalaa ja epämiellyttävää	Kiinnitetään käytettävyyteen huomiota uuden ohjelman suunnittelussa
Ohjainohjelma ja PLC ovat riippuvaisia yhdestä henkilöstä	Ongelmatilanteissa kukaan muu ei voi auttaa eikä korjata, sillä kukaan muu ei ymmärrä järjestelmää	Tutustutaan järjestelmään ja tehdään uuden ohjelman ohjelmakoodista selkeä, ymmärrettävä ja hyvin dokumentoitu
Tulevaisuudessa PLC:n ja tietokoneen käyttämät sarjaportit korvataan paikallisverkolla	Oleellinen osa uudesta ohjainohjelmasta tulee vanhentumaan joissakin vuosissa	Huomioidaan tämä uuden ohjelman suunnittelussa, jotta vain osa siitä täytyy korvata tulevaisuudessa

