Marko Wegelius

Konfiguraatiotyökalun suunnittelu ja toteutus

Tieto- ja viestintätekniikan insinööri

peliteknologia ja älykkäät järjestelmät

Syksy 2023

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Wegelius Marko Jaakko

Työn nimi: Konfiguraatiotyökalu

Tutkintonimike: Tieto- ja viestintätekniikan insinööri (AMK)

Asiasanat: Konfiguraatio, .Net Framework, Enviset, Parsetus

Opinnäytetyön aiheena oli luoda konfiguraatiotyökalu asiakasohjelmiston tarpeisiin. Tavoitteena oli selvittää ohjelman vaatimukset, suunnitella ohjelman rakenne ja toiminta, ohjelmoida ohjelma ja testata luodun ohjelman toiminta asiakasympäristössä.

Tavoitteena oli myös luoda ohjelman kaikki tarvittavat dokumentaatiot. Ohjelma oli tarkoitus ottaa käyttöön osana asiakasohjelmistoa, jolloin se olisi osa samaa kokonaisuutta.

Tavoitteena oli myös käyttää annettuja ohjelmistostandardeja ja tutkia miten ohjelmiston elinkaari saadaan turvattua tulevaisuuden tarpeita ajatellen.
Työn tuloksena saatiin ohjelmisto, joka vastaa sille annettuja kriteerejä, joskin nykyiset vaatimukset ovat muuttuneet. Ohjelmisto toimii asiakasympäristössä ja sen muokkaaminen tulevaisuutta varten on kohtalaisen yksinkertaista, mikä oli yksi työn päätavoitteista.

Ohjelmisto vastaa asiakasyrityksen sisäisiä sekä yleisesti hyväksi katsottuja standardeja ohjelmistokoodin puolesta. Ohjelma testattiin käyttäen manuaalitestausta, jonka ohjelma läpäisi.

Toteutuksessa onnistuttiin työn vaativuuteen nähden hyvin. Ohjelma toimii ympäristössään ja tuottaa halutun palvelun käyttäjälle, joskin muuttuneiden vaatimusten takia ohjelmistoa joudutaan vielä kehittämään lisää. Tästä syystä ohjelmiston käyttöönotto viivästyy.

KAMK • University of Applied Sciences

Abstract

Author(s): Wegelius Marko Jaakko

Title of the Publication: Design and Implementation of a Configuration Tool

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: configuration, .Net Framework, Enviset, Parsing

The subject of the thesis was to create a configuration tool for the needs of the client's software. The goal was to find out the program's requirements, plan the program's structure and operation, code the program and test the operation of the created program in the client's environment.

A further goal was to create all the necessary documentation for the program. The program was supposed to be introduced as part of the client's software in which case it would be part of the same entity. The goal was also to use the given software standards and to study how the life cycle of the software can be secured with future needs in mind.

The result of the work was software that meets the given criteria, although the current requirements have changed. The software works in a customer environment and modifying it for the future is relatively simple, which was one of the main goals of the work.

The software corresponds to the client company's internal and generally accepted standards for the software code. The program was tested using manual testing, which the program passed. The implementation was successful considering the complexity of the work. The program worked in its environment and produced the desired service for the user, although due to the changed requirements, the software still needs to be developed further. For this reason, the implementation of the software is delayed.

Sisällys

1 Johdanto

Viimeisen puolentoista vuoden aikana olen työskennellyt lentävän kaluston teknisen datan purku- ja arkistointipalvelimen asennukseen sekä testaukseen liittyvissä tehtävissä toimeksiantajan palveluksessa. Työn edetessä on havaittu erilaisia konfiguraatiokombinaatioita olevan suuri määrä, ja kombinaatioiden määrä tulevaisuudessa on kasvava. Konfiguraatioiden ja niiden eri kombinaatioiden hallinta on todettu manuaalisesti erittäin haastavaksi. Eri konfiguraatioiden ja niiden kombinaatioiden hallinta manuaalisesti kasvattaa inhimillisten virheiden riskiä ja on aikaa vievää.

Lopputyössäni kehitetään havaittuun ongelmaan ratkaisu. Opinnäytetyössäni suunnitellaan ja toteutetaan konfiguraatiotyökalu lentävän kaluston teknisen datan purku- ja arkistointipalvelimen tarpeeseen.

Konfiguraatio-ohjelmistolla hallitaan palvelinympäristön konfiguraation eheyttä keskitetysti. Keskitettävyys tässä tapauksessa tarkoittaa, että asiakasjärjestelmäohjelmiston kaikki konfigurointitiedostot ovat hallittavissa samasta paikasta.

Palvelinohjelmistojen asentamisen jälkeen ympäristö tulee konfiguroida käyttökohteen vaatimaan rooliin. Konfiguraation eheys tulee tässä vaiheessa tarkastaa. Käytännössä konfigurointi konfiguraatio-ohjelmalla tarkoittaa erilaisten asetustiedostojen, asetus-tietokantataulujen, rekisterin sekä scriptien sijaintien tarkastamista ja tiedostojen sisällön oikeellisuuden tarkastamista (parsettamista). Konfiguraatio-ohjelmiston käyttöliittymä kertoo tarkastuksen tuloksen palvelimen ylläpidolle.

Konfiguraatio-ohjelmiston käyttöliittymä on dynaamisesti muuntuva. Sen konfiguraatiotiedostossa määritellään tarkastettavat kohteet, jolloin ohjelmiston tarkastamat kohteet sekä käyttöliittymä mukautuvat tilanteeseen dynaamisesti.

Konfiguraatio-ohjelmiston päätarkoitus on nopeuttaa sekä laadullisesti kehittää palvelinympäristön käyttöönottoa sen eri rooleihin. Lisäksi tarkoituksena on minimoida inhimillisten virheiden mahdollisuus minimiin.

Kiinnostavan työstä teki se, että tämän ohjelmiston dynaaminen arkkitehtuuri on hyödynnettävissä pienillä muutoksilla myös muussa ohjelmointityössä, jolloin voidaan tulevaisuudessa kehittää ohjelmistoratkaisuja, joita ei tarvitse niin useasti päivittää.

Ohjelmisto päätettiin rajata siten, että sen tulisi vain käsitellä tekstipohjaiset konfiguraatiotiedostot sekä joitakin ohjelmistotarkastuksia rekisteristä eikä esimerkiksi Active Directory -ympäristön konfiguraatioita.

2 Ohjelmistoarkkitehtuuri

Kirjassa Game Engine Architecture käydään läpi pelimoottorin arkkitehtuuria, jota voitiin myös hyödyntää tässä työssä, sillä kuten pelimoottorisuunnittelussa ja -arkkitehtuurissa on loppujen lopuksi kyse ohjelmistosuunnittelusta. [1, s. 105.]

2.1 Luokka

Olio-orientoitunut ohjelmointi rakentuu luokasta ”Class”, joka koostuu attribuuteista eli datasta sekä käytänteistä eli koodista, jotka yhdessä muodostavat käytännöllisen kokonaisuuden. Luokka on siis spesifikaatinen kuvaus siitä, miten ohjelmassa toimivat objektit tulisi rakentaa. [1, s. 106.]

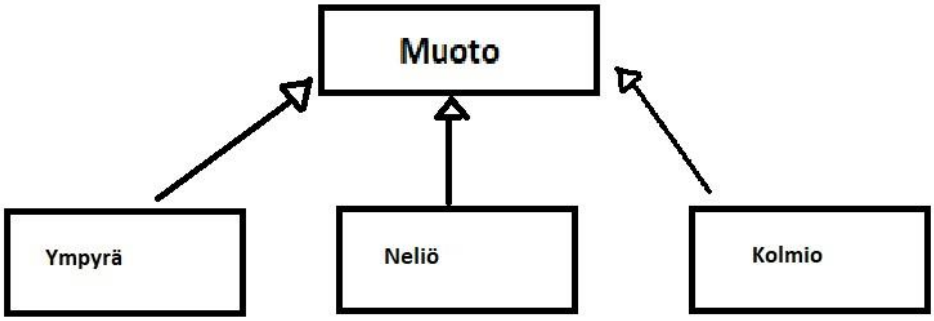
2.2 Kapselointi

Kapselointi tarkoittaa yhteen kuuluvien tietojen ja toimintojen kokoamista yhdeksi kokonaisuudeksi. Olio-ohjelmoinnissa tämä tapahtuu kokoamalla muuttujat eli data sekä toiminnallisuusfunktiot luokaksi. Kapseloinnissa myös piilotetaan olion sisäinen toteutus käyttäjältä. Tällä on siis tarkoitus tarjota käyttäjälle eräänlainen valmis rakennuspalikka, jonka sisäisestä toiminnasta käyttäjän ei tarvitsisi välittää. [1, s. 106.]

2.3 Perintä

Perinnällä voi tehdä niin sanotun lisäosaluokan vanhasta luokasta, jolloin voi käyttää vanhaa koodia uudelleen ja eri tarkoituksiin esimerkiksi luokasta muoto voi perinnän kautta tarkentaa luokan ympyrään, neliöön tai kolmioon. [1, s. 106.]

Perintä on hierarkkista. Kuten kuvassa 1 näkyy, pääluokka on muoto, josta peritään lapsi- luokat ympyrä neliö ja kolmio. Nämä lapsiluokat saavat näin käyttöönsä muotoluokan funktiot sekä attribuutit sekä omien luokkiensa tarkentavat funktiot ja attribuutit. [1, s. 107.]



Kuva 1. Perintähierarkia [1, s. 107.]

2.4 Ohjelmointikielten ominaisuudet

Kirjassa mainitaan hyvä esimerkki, mitä saattaa tapahtua, kun käytetään uusimpia kielten ominaisuuksia. Kirjan kirjoittajan yrityksessä suhtauduttiin hyvin varovaisesti uusimpiin ominaisuuksiin, vaikka ne tehostaisivat työtä. Esimerkkinä annettiin ”auto”-tyyppi C++-kielessä sama ominaisuus löytyy myös C#-kielestä ja se on ”var”-tyyppi. Auto-tyyppi antaa C++-kielessä automaattisesti oikean tyypin muuttujalle esimerkiksi luku 7.5 saa float- tai double-tyypin muuttujan käännösvaiheessa. Tämä ominaisuus vie tyypitetyn kielen hyödyt pois ja kielenluku vaikeutuu, kun lukija ei heti tiedä, mitä esimerkiksi funktion pitäisi palauttaa. Tämän takia tässä opinnäytetyössä on kaikki muuttujat merkitty oikeilla tyypeillään eikä ole käytetty yhtään var:ria. [1, s. 117.]

2.5 Ohjelmointistandardit

Ohjelmointistandardit ovat ohjeita, joilla määritetään ohjelmistoprojektissa käytettävät menetelmät ja tuotetun lähdekoodin ulkoasu sekä rakenne. Tavoitteena on tuottaa helpommin luettavaa, sekä virheettömämpää tekstiä, jota on helpompi ylläpitää tulevaisuudessa. On olemassa yleinen ymmärrys siitä mitä ohjelmiston tulisi saavuttaa, jotta sen laatua voitaisiin kuvata hyväksi. [2.]

Ian Sommerville on esimerkiksi määritellyt seuraavat neljä kohtaa:

2.5.1 Ylläpidettävyys

Ohjelmiston kehitettävyys, muunneltavuus, tekninen velka ja ohjelmistokoodin luettavuus. [3.]

2.5.2 Luotettavuus

Esimerkiksi ohjelmiston tarjoamaan palvelun toimintaan ja ylläpitoon voidaan luottaa sovitulla ajanjaksolla, eikä se vuoda käyttäjien henkilökohtaisia tietoja ulkopuolisille, vaikka olisikin hyökkäyksen kohteena. [4.]

2.5.3 Tehokkuus

Esimerkiksi ohjelmistokoodissa tehtävä järkevä roskienkeruu, jolla vapautetaan järjestelmän resursseja silloin, kun niitä prosesseja ei enää tarvita. Toinen esimerkki on myös käyttää oikeanlaisia algoritmeja suurien datamassojen kanssa. [1.]

2.5.4 Käytettävyys

Ohjelmistolla tehtäväksi tarkoitetut työt voidaan suorittaa laadukkaasti, tehokkaasti ja helposti, eikä ohjelmisto aiheuta käyttäjälle enemmän harmia kuin hyötyä. Esimerkiksi jos ohjelman tarjoama yleisesti käytetty työkalu löytyy neljän alivalikon takaa, eikä siihen ole mahdollista käyttää pikanäppäintä voidaan sanoa, että käytettävyys on heikko kyseisellä työkalulla. [5]

Gerald Weinberg on määritellyt neljä kysymystä, joiden kautta ohjelman hyvyyttä voidaan arvioida:

- Täyttääkö ohjelmisto sille asetetut vaatimukset?

- Onko ohjelmisto aikataulussa sekä budjetin asettamissa rajoissa?
- Voiko ohjelmistoa muuttaa tulevaisuudessa muuttuvassa ympäristössä?
- Onko ohjelmisto tehokas ympäristössä sekä tehtävässä, johon se on suunniteltu? [1.]

2.5.5 Esivalmistelut

Ennen kuin ohjelmointi aloitetaan, tulisi selvittää, onko kaikki tarvittavat tiedot koottu. Mikäli tarvittavia taustatietoja ei ole tai ne ovat pahasti vajavaiset, voidaan jo ennen ohjelmoinnin aloittamista olettaa, että ohjelmisto ei tule täyttämään laadullisia vaateita. Wikipedian sivulla mainittiin seuraavat kysymykset, joihin tulisi vastata ennen kuin ohjelmointi aloitetaan [1]

- Miten ohjelmiston kehitys on suunniteltu? (elinkaari)
- Mitä ohjelmiston tulisi tehdä? (vaatimusmäärittely)
- Mikä on ohjelmistojärjestelmän kokonaissuunnitelma? (arkkitehtuuri)
- Mitkä ovat tarkat määritykset ohjelmiston erikomponenteille? (suunnittelu)
- Mitä ohjelmistokieltä käytetään?

2.5.6 Standardit tässä työssä

Ohjelmointistandardit ovat olemassa tässä työssä lähinnä kahdesta pääsyystä

- Ne tekevät koodista luettavampaa, ymmärrettävämpää ja yläpidon kannalta parempaa, esimerkiksi standardissa voidaan raja, mitä eri frameworkeja saa koodissa käyttää.
- Ne estävät ohjelmoijia tekemästä helposti vältettäviä virheitä tulevaisuutta ajatellen, esimerkiksi suosimalla helpommin luettavia ominaisuuksia sekä käyttämällä pienempiä helpommin testattavissa olevia kirjastoja.

Yleisemmin voidaan sanoa, että hyvä standardi pitää sisällään ohjeet kommentoinnista, että funktioiden, attribuuttien sekä luokkien nimeämisestä.

3 Suunnitelma työn tekemiseksi

Aikaisempi asennusohjelmisto aiheutti valtavia riskejä, sillä asennettavan ohjelmiston kriittisyyden takia sen asennus ja tietokantaympäristö eivät saaneet missään tilanteessa epäonnistua. Ohjelmisto kaikkine lisäosineen ja tietokantarakenteineen oli hankala hallittava ja tarvitsi useita erillisiä konfiguraatioita johtuen tuotteiden eri valmistajista ja siitä, että nämä ohjelmat piti jollain tavalla saada kommunikoimaan keskenään samassa järjestelmässä. Yhtenä ainoana yhdistävänä tekijänä näillä ohjelmilla oli se, että niiden konfiguraatiotiedostot olivat selkokielisiä tekstitiedostoja, jolloin pystyttiin tekemään erillinen ohjelmisto, jonka tehtävä oli yhdistää ja hallita näiden ohjelmistojen konfiguraatioita ja näin turvata asennusvarmuus.

Ennen kuin oli erillistä konfiguraatio-ohjelmistoa nämä asennukset tehtiin käsin, mikä aiheutti sen, että asennusta jouduttiin harjoittelemaan useita kertoja laboratorio-oloissa. Tämä lisäsi kustannuksia merkittävästä ja aiheutti tekijöissä jatkuvaa turhautumista sekä työmäärä oli merkittävästi suurempi kuin se olisi, jos konfiguraatiot voitaisiin tehdä automaattisesti.

Konfiguraatio-ohjelmistolla voitiin varmistaa, että asennettavan ohjelman toiminta ympäristössään oli vaatimusten mukaista ja turvallista, sillä valmiiksi laboratoriossa testatut konfiguraatiot saatiin siirrettyä suoraan käyttöön ilman ihmisen tuomaa epävarmuutta.

Työn tarve ilmeni tehdessä asiakasohjelmiston asennusta ja siinä huomatuista vajavaisuuksista johtuvista korjaustarpeista. Tämä asennus oli tyypiltään sellainen, jossa ei saanut tapahtua virheitä, jotenka päätettiin tehdä työkalu, jolla voitaisiin asennus suorittaa varmennetulla tavalla.

Työtä lähdettiin suunnittelemaan selvittämällä tarpeet ja metodit, jolla saataisiin aikaan mahdollisimman helposti ja nopeasti työkaluohjelmisto, jonka vuoksi ei tarvitsisi kääntää asiakasohjelmistoa uudelleen ja näin säästettäisiin aikaa ja resursseja.

Koska suurimmat ongelmat huomattiin liittyvän asiakasohjelmiston konfigurointitiedostoihin sekä vaatimuksiin muista tarvittavista ohjelmistoista sekä niiden versioista. Päätettiin tehdä uusi ulkoinen ohjelmisto, jonka tehtäväksi tulisi näiden tiedostojen sekä ohjelmistojen olemassaolon tarkistaminen sekä editointi.

3.1 Työn suunnittelun vaiheet

Heti aluksi saatiin työlle vaatimus siitä, että toteutettavan ohjelmiston tulisi toimia dynaamisesti, jolloin ohjelmistoa voisi tulevaisuudessa käyttää myös muihin asiakasohjelmistoihin, joissa on samantyyppisiä ongelmia.

Ohjelman vaatimus dynaamisuudesta suuntasi käyttämään ohjelmointitekniikoita, joista ei aikaisemmin ollut kokemusta eikä ymmärrystä. Tämän takia jouduttiin konsultoimaan useita henkilöitä eri projekteista, joilta saatiin tarvittavia ohjeita sekä oppimateriaalia. Nämä materiaalit olivat suureksi osaksi luokiteltuja. Käsiteltävän materiaalin sekä toimintaympäristön takia jouduttiin myös tekemään suunnittelua omaan kokemukseen sekä ajatteluun perustuen.

Termistöllä lähdemateriaaleissa yleensä tarkoitetaan eri asiaa kuin tässä työssä. Tämän sekä aikataulukiireiden takia jouduttiin turvautumaan omaan kokemukseen sekä pohdintaan, miten ohjelmiston rakenne toteutettaisiin toimivasti.

Ohjelman dynaamisuus tässä yhteydessä tarkoitti, että ohjelma pystyy rakentamaan ”itsensä” ilman, että lähdekoodiin koskettaisiin. Tämän sai aikaan sen, että ohjelmaan luotiin oma init-tiedosto, jossa tarvittavat polut sekä tiedostojen ja ohjelmistojen nimet sijaitsevat. Tästä tiedostosta saatiin parsittua ohjelmallisesti tiedot, minkä jälkeen ohjelma rakensi käyttöliittymän tarvittavien tietojen mukaan.

3.2 Käytettävät työkalut

Työ suunniteltiin käyttäen hyväksi sivulla https://learn.microsoft.com/ olevaa C# .Net framework -materiaalia, jolla selvitettiin, mitä valmiita komponentteja olisi tarjolla, jolla työtä saataisiin nopeutettua. [6.]

Täältä esimerkiksi otimme komennon ”Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location”, jolla saatiin konfiguraatio-ohjelmiston suorittavan exe-tiedoston sijainti, jonka ympärille päästiin ohjelmiston tiedostorakennepuu rakentamaan.

Työn ohjelmointiin käytettiin Microsoft Visual studio -2016 versiota ja .Net framework -versiota 4.7.2 Testaukseen käytettiin manuaalitestausta, koska ohjelmiston koko oli suppea ja testitikettien määrä pieni.

Manuaalinen testaus on ohjelmistotestausta, missä testit tehdään ohjelmistolle ilman automaatiotyökaluja. Testaaja siis käyttää ohjelmistoa annetuiden testitapausten mukaisesti, jotka ohjelmoija on aiemmin määritellyt ja jotka tarkistavat tuloksen. Tämän jälkeen tuloksista kirjoitetaan raportti, jonka pohjalta ohjelmoijat voivat korjata ohjelmistoa.

Manuaalitestaus on siis alkeellisin ohjelmistotestauksen muoto ja käytetään yleisesti ohjelmistovirheiden etsimiseen. Manuaalinen testaaminen vaatii enemmän vaivaa kuin automaattitestaus, mutta se on välttämätöntä ohjelmiston toiminnan varmistamiseksi. Etuna manuaalitestauksessa on, että ei tarvitse erillisiä testausohjelmistoja ja on siksi edullisempi ratkaisu. Automaattinen testaus ei myöskään ole aina 100 % mahdollista. [7.]

4 Vaatimusmäärittely

Ohjelmiston vaatimusmäärittely tehtiin vastaamaan asiakasohjelmiston tarpeita ja noudattaen asiakasyrityksen ohjelmointikäytänteitä ja -standardeja.

Ohjelmiston tuli käynnistyessään tarkastaa ohjelmiston oman executiven sijainti ja onko sen sijainnin kansiorakenteen juuressa kansiot nimeltään ”Archives” ja ”Temp” sekä tiedosto config.ini.

Mikäli Archives- tai Temp-kansiot puuttuvat, ohjelmisto luo ne.

Archives-kansion sisään tulee varmuuskopiokansiot, joiden sisään tulee tiedostot jotka, ovat FilePath.ini:ssä mainittu. Archives-kansion sisäkansiot saavat nimensä sen hetkisestä kellonajasta sekä päivämäärästä, milloin varmuuskopio on luotu.

Varmuuskopiot otetaan aina ohjelmiston käynnistyessä.

Temp-kansioon tulee myös samat tiedostot kuin varmuuskointiin, mutta niitä pystyy käyttöliittymässä editoimaan.

Config.ini-tiedosto sisältää raakatekstinä haluttujen tiedostojen nimet sekä polut, mistä tiedostot kopioidaan sekä varmuuskopintiin sekä temp-kansioon. FilePath.ini-tiedoston polkurakenne toimii esimerkin mukaisesti:

```
”[InstalFiles]
  HostFile=C:\Windows\System32\drivers\etc\hosts
  [ProgramPaths]
  Excel=Software\Microsoft\Office\14.0\Excel\ExcelName”
```

[InstalFiles] on tagi merkintä, joka kertoo konfiguraatio-ohjelmalle, mitä tämän alla oleville tiedoille tulee tehdä.

”HostFile” on nimi, joka kertoo konfiguraatio-ohjelmistolle, mitä käyttöliittymän nimi kentässä tulee esittää.

”C:\Windows\System32\drivers\etc\hosts” on polku, missä tiedosto sijaitsee.

[InstalFiles] ja [ProgramPaths] ero on siinä, että [ProgramPaths] tagin alla olevat tiedostot etsitään rekisteristä ja ne sisältävät tiedostopolun sijaan rekisteripolun.

Yhtäsuuruusmerkki (=) on tekstin parsinnan helpottamista varten, jolla tiedot erotetaan toisistaan.

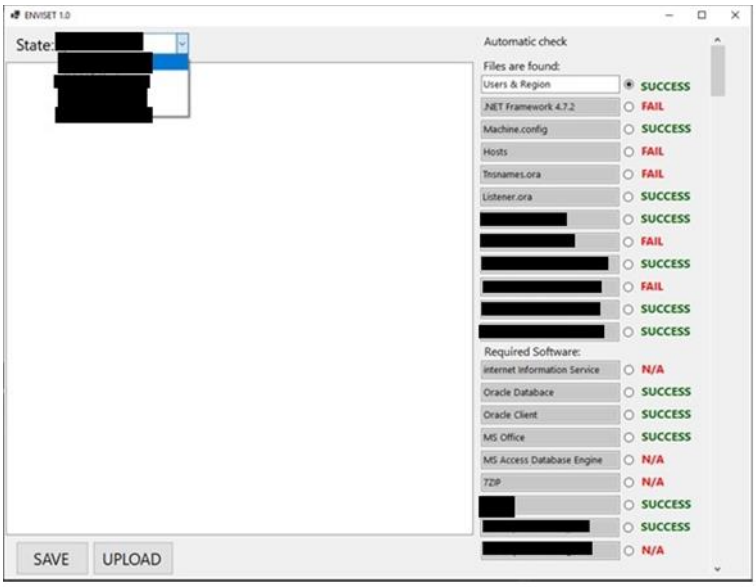
Tällä tavalla tehtynä saatiin ohjelmisto luotua nopealla aikataululla sekä sen uudelleen käytettävä ohjelmisto myös tulevaisuudessa, jos niin halutaan. Kun halutaan lisätä polkujärjestelmään, se vain lisätään FilePath.ini-tiedostoon, jolloin ohjelmiston uudelleen kääntämistä ei tarvita eikä tarvitse tehdä muutoksia vaadittaviin asiakirjoihin ja versionumeroihin, mitkä vievät aikaa ja resursseja.

Ohjelma tarkastaa myös, että rekisteristä on määritetyt ohjelmistot asennettu ja palauttaa arvon ”True”, jos on.

Kun FilePath.ini-tiedoston luku sekä varmuuskopiointi on tehty, luodaan käyttöliittymänäkymä tarvittavista komponenteista. Komponenttien asetukset määräytyvät tietueen tagin mukaisesti. Esimerkiksi asennetun ohjelmiston executivea ei voida editoida, jotenka se ei myöskään saa tarvittavaa radionappia siihen.

kuvan 2 mukaisesti ohjelma rakentaa Config.ini-tiedostosta parsituista tiedosta käyttöliittymän, jossa on nimi kenttä, radionappi sekä kenttä tiedolle, jossa näkyy, onko tiedostoa löytynyt.

Mikäli tiedostoa ei löydy, tulee teksti ”FAIL” tai ”N/A” punaisella värillä ja jos tiedosto löytyy, tulee teksti ”SUCCESS” vihreällä värillä.



Kuva 2. Konfiguraatio-ohjelman käyttöliittymä [8]

Editoitavat tiedostot ja ohjelmien executivet on eroteltu toisistaan otsikoilla ”Files are found” ja ”Required software”.

Painamalla radionappia avautuu editoitava tiedosto, joka on kopioitu Config.ini tiedoston sisältämästä polusta Enviset-ohjelmiston temp-kansioon. Tiedosto avautuu tekstimuodossa tekstieditoriin (iso valkoinen pohja vasemmalla kuvassa). Tässä editorissa tekstiä voi muokata haluamallaan tavalla ja muokatun tiedoston voi tallentaa temp-kansioon painamalla ”SAVE”-nappia, jolloin voi mennä muokkaamaan jotain muuta haluamansa tiedostoa. Tämä siksi, että mikäli tekee muokkaukset tiedostoon ja poistuu tekemään jotain muuta tiedostoa painamatta ”SAVE”-nappia tiedostoon tehdyt muokkaukset eivät tallennu.

”UPPLOAD”-nappi lähettää editoidun tiedoston alkuperäiseen polkuunsa, mikä löytyy Config.ini-tiedostosta. ”UPPLOAD”-nappia painettaessa ohjelmisto kysyy, oletko nyt aivan varma, että haluat lähettää tiedoston, mikäli käyttäjä painaa kyllä, ohjelmisto korvaa alkuperäisen tiedoston editoidulla tiedostolla.

”STATE”-valikko on tehty sitä varten, että mikäli halutaan ajaa useita eri Config.ini-tiedostoja, joissa on eri polut. Näin voidaan käyttää ohjelmaa asentaessa eri spesifikaatioilla olevien asiakasohjelmistojen asennuksessa.

Ohjelmiston rakenne haluttiin saada mahdollisimman yksinkertaiseksi ja käyttöliittymä nopeasti hahmotettavaksi, jotta sen oppimiskynnys olisi mahdollisimman matala. Näin säästytään koulutustarpeelta.

Ohjelman tulee kerätä erilliseen lokitiedostoon virheilmoitukset, jotka ovat muodoltaan asiakasyrityksen standardien kaltaiset. Loki-ilmoituksia kerätään kolmen erityyppistä: varoitus, informaatio ja virhe. Varoitus tulee esimerkiksi siitä, että Config.ini-tiedostossa mainittuja polkuja ei löydy. Virhe tulee siitä, että ohjelma suorittaa jonkun laittoman toiminnon ja esimerkiksi kaatuu.

5 Konfiguraatio-ohjelmiston ohjelmointi

Työ aloitettiin kartoittamalla siitä, oliko asiakasyrityksessä tehty aikaisemmin vastaavaa ja saisiko siitä otettua mallia. Joitakin vastaavanlaisia ohjelmistoja löytyi ja niiden lähdekoodista katsottiin olevan hyötyä tarkastelussa ja arkkitehtuurissa, mutta koska prosessi niiden suoraan käyttöön aiheutti kohtuutonta vaivaa, niin päätettiin, että niitä käytetään vaan suuntaa antavina. Tämän takia koko ohjelmiston koodi suunniteltiin koko kokonaisuudessaan uutena ja vain käyttämällä asiakasyrityksen sisäisiä ohjelmointistandardeja harkitusti.

Sisäisistä ohjeista voidaan antaa esimerkiksi tapa, jolla asiakasyrityksessä tuli kommentoida koodia

```
//-------------------------------------------------------------------------------------------
//                  Asiakas yrityksen nimi
//-------------------------------------------------------------------------------------------
// SOFTWARE:  (Ohjelmisto)
// SYSTEM:       (Järjestelmä)
// SUBSYSTEM: (Alijärjestelmä)
// FILE:          (Tiedoston nimi)
// REFERENCE:  (Mahdolliset viitteet)
-------------------------------------------------------------------------------------------
// DESCRIPTION:        (Tiedoston kuvaus)
// NOTES:                 (Huomautukset, rajoitukset, yms. lisätiedot)
-------------------------------------------------------------------------------------------
// MODIFICATIONS:
//
// VERSION:              (Versio numero)
// DATE:                    (Päivämäärä)
// AUTHOR:               (Tekijä)
// TOOLS:                  (Kääntäjän tai IDE-työkalun versio=
// DESCRIPTION:        (Muutoksen kuvaus)
//
//-------------------------------------------------------------------------------------------
```

Edellisen sivun pohjassa ”*Muutosten yhteenvetotiedot merkitään Modifications-osaan laskevassa järjestyksessä niin, että viimeisin muutos (se mikä ensimmäisenä kiinnostaa) on aina ensimmäisenä*” (lainaus tekninen ohje). Myös funktioiden tiedot komentoitiin seuraavasti:

```
//-------------------------------------------------------------------------------------------
// TITLE:                (funktion nimi)
// DESCRIPTION:          (funktion kuvaus)
// INPUTS:               (arvot, joita funktio mahdollisesti saa)
// OUTPUTS:              (arvot, joita funktio mahdollisesti palauttaa)
// NOTES:                (rajoitukset tai muut erityishuomiot)
//-------------------------------------------------------------------------------------------
```

Harkinnanvaraisuus asiakasyrityksen standardia noudattaessa tuli siitä, että ohjeistukset oli luotu C++-kielen käyttöön ja koska työssä käytettiin C#-kieltä, jouduttiin joitakin kohtia standardissa soveltamaan.

Koska ohjelmiston vaatimuksissa mainittiin dynaamisuusvelvoite, niin päätettiin ensiksi tehdä tiedosto, johon halutut nimet ja polut koottaisiin. Tiedoston rakenne päätettiin tehdä seuraavan laisesti:

```
”[InstalFiles]
HostFile=C:\Windows\System32\drivers\etc\hosts
[ProgramPaths]
Excel=Software\Microsoft\Office\14.0\Excel\ExcelName”
```

[InstalFiles] on tagi, joka kertoo ohjelmalle, mitä tämän alla oleville tiedoille tulee tehdä. ”HostFile” on esimerkki tiedoston nimi, joka kertoo ohjelmistolle, mitä käyttöliittymän nimi kentässä tulee lukea. ”C:\Windows\System32\drivers\etc\hosts” on polku, missä esimerkkitiedosto sijaitsee kansiorakenteessa. ”Software\Microsoft\Office\14.0\Excel\ExcelName” on taas rekisteripolku.

[InstalFiles] ja [ProgramPaths] ero on siinä, että [ProgramPaths]-merkinnän alla olevat tiedostot etsitään rekisteristä ja ne sisältävät tiedostopolun sijaan rekisteripolun.
Yhtäsuuruusmerkki (=) on parsijaa varten oleva merkintä, jolla tiedot erotetaan toisistaan.
Tämän tiedoston sisäisen rakenteen ympärille alettiin valmistamaan parsijaa, jonka tehtäväksi tuli erotella tekstitiedoston sisältä osat InitList-nimiseen julkiseen luokan olioon, joka sijaitsee

InitLoader.cs-tiedostossa. Näitä osia olivat ”FileName” eli tiedoston nimi, ”FilePath” eli tiedoston alkuperäinen polku, ”FileTag” eli joko [InstalFiles] tai [ProgramPaths] riippuen siitä, minkä alla kyseinen tieto sijaitsi sekä ”FileTempPath”, joka oli tiedoston polku Enviset-ohjelmiston sisäisessä temp-kansiossa.
Tämän jälkeen parsitut tiedot InitList-luokassa laitettiin julkiseen listaluokkaan iniparts, josta niitä pystyi myöhemmin muualla kutsumaan.

5.1 Kansioiden luonti sekä tiedoston käsittely

DirectoryHandler.cs on tiedosto, jossa sijaitsee DirectoryHander-luokka, jonka tehtävänä on luoda temp- ja archives-kansiot konfiguraatio-ohjelman ensimmäisessä käynnistyksessä ja tarkastaa, onko nämä kansiot luotu. Kansiot luodaan konfiguraatio-ohjelman executiven viereen. Temp-kansioon tulee kopiot alkuperäisistä tiedostoista, joita kansion sisällä voi muuttaa konfiguraatio-ohjelmalla. Archives-kansion sisään taas alkaa rakentua kansiopuu, jossa on varmuuskopiot alkuperäisistä tiedostoista, jotka myös temp-kansioon laitettiin. Varmuuskopiokansion nimi muodostuu päivämäärästä ja kellon ajasta, jolloin niitä ei tule koskaan kahta samanlaista. Tällä hetkellä konfiguraatio-ohjelma tekee aina käynnistyessään uuden varmuuskopion alkuperäisistä tiedostoista automaattisesti, mutta tulevaisuudessa tätä ominaisuutta saatetaan vielä muuttaa.

FileHandler.cs
FileHandler.cs on tiedosto, jossa sijaitsee FileHandler-luokka. FileHandler-luokan tehtävä on kopioida InitLoader-inipartslista olion polkutietoja temp, archives ja alkuperäiseen polkuun eri funktioilla.

CopyFile funktio esimerkiksi saa InitLoader-tyypin muuttujan, joka sisältää tiedot Config.ini-tiedostossa parsitussa muodossa sekä indeksinumeron. Initloader-tyypin muuttujasta otetaan FilePath-tieto, josta funktion sisällä tämän jälkeen parsitaan vielä erikseen polku File.Copy-funktiota varten. Erottelu tapahtuu viimeisestä ”\” merkistä, jolloin jäljelle jää vain puhdas polku eikä tiedoston nimeä ole polussa mukana. CopyFile-funktion tämänhetkinen päämäärä polku on konfiguraatio-ohjelman temp-kansio.

UploadFile-funktio taas toimii samalla tavalla kuin copyFile-funktio, mutta sen päämäärä on toisin päin eli se lähettää temp-kansiossa olevat tiedostot InitLoader iniparts -listan polkuihin ja korvaa siellä olevat tiedostot.

CopyFile-funktiota on myös yliajettu toisella samannimisellä, jossa funktio saa lisämuuttujan string directoryPath, jolla voi määrätä kopioitavan tiedoston päämäärä polun.

FileHandler-luokassa on myös funktiot FileExists, joka tarkistaa polussa halutun nimistä tiedostoa ja jos on niin funktio palauttaa ”True”. Sekä funktio GetProgramRec, joka tarkistaa rekisterissä InitLoader-inipartslistaolion polkuja, joissa on merkintä [ProgramPaths] ja jos se löytää ne palauttaa funktio ”True”.

5.2 Lokin kirjoitusfunktion toiminta

LogInfo.cs

LogInfo.cs tiedosto sisältää luokan LogInfo, jonka tehtävänä luoda log.txt-tiedosto konfiguraatio-ohjelman viereen sekä hoitaa log.txt-tiedoston kirjoitus. Tähän tiedostoon tulee kaikki konfiguraatio-ohjelman viestit kronologisessa järjestyksessä. Log.txt-tiedostossa on aina ensimmäisenä teksti, jossa näkyy ohjelmiston nimi sekä versio. Login tekstit kirjoitetaan kutsumalla appenLog-funktiota ja syöttämällä siihen haluttu teksti, funktio lisää tekstin eteen aikaleiman ja tämän perään itse syötetyn tekstin sekä seuraavalle riville tekstin erotusviivan.

Form1.cs

Tässä tiedostossa luotiin aluksi luokkaolio InitLoader loader, FileHandler makeFile, DirectoryHander backupDir. Lista muuttujat UiLabels, joka oli tyyppiä Label, UiNameBox tyypiltään TextBox, UiCheckButton tyypiltään Radiobutton ja UiHeaderLabel, joka oli tyyppiä Label. Sekä sisäiset muuttujat UiTextBox tyypiltään Textbox, UiSaveTempButton tyypiltään Button, UiUploadButton tyypiltään Button sekä julkiset muuttujat inipartsIndex tyypiltään int ja tempDir, joka sai arvokseen ohjelmiston exen sijainnin sekä ohjelmiston temp-kansion polun sijainnissa.

Formin kooksi alustettiin 1024, 768 pixeliä Size-komennolla. Formi sai myös yläpalkkiin versionumeron sekä tuotenimen, myös tarvittavat funktiot kutsuttiin load-funktiossa, jolloin ne ajettiin käynnistyksen yhteydessä.

CheckFiles-funktion tehtävänä on luoda FileParth.ini-tiedostosta parsituista tiedoista käyttöliittymäelementit kutsumalla CreateLabel-, CreateNameBox- ja CreateCheckButton-funktioita. Funktiot saavat InitLoader-luokka oliosta tarvitsemansa tiedot luodakseen käyttöliittymäelementit. Käyttöliittymäelementit saadaan näin tehtynä olemaan dynaamiset, jolloin ylläpito helpottuu. Funktion myös kopioi tiedostot konfiguraatio-ohjelmiston omiin temp- ja achives-kansioihin, mikäli ne on merkitty ”InstalFiles” tagillä sekä funktio kutsuu GetProgramRec funktiota tarkistaakseen, löytyvätkö asennetut ohjelmistot rekisteristä.

5.3 Käyttöliittymän save-napin toiminta

CreateUiSaveTempButton-funktion tarkoitus on luoda kutsuttaessa käyttöliittymäelementti, joka toimii temp-kansiossa olevien tiedostojen muutosten tallentamisessa. Funktio on sidottu UiSaveTempButton_Click-funktioon eventtinä.

Muutokset voi tehdä käyttöliittymässä olevaan tekstikenttään, mutta mikäli tiedoston sulkee tai vaihtaa toiseen, eivät muutokset tallennu automaattisesti. Tämä siksi, että mikäli käyttäjä teki jotain, mitä ei halunnut, voidaan näin palauttaa muutokset takaisin alkuperäiseen tilaan vain avaamalla jokin toinen tiedosto valikosta ja palaamalla takaisin. Mikäli kaikki halutut muutokset on tehty ja ne ovat tarkistettu, voidaan painaa ”SAVE”-nappia ja nämä muutokset jäävät muistiin temp-kansiossa olevaan tiedostoon. Funktio käyttää hyväkseen .Net-frameworkin StreamWriter-luokkaa, jolla muutokset kirjoitetaan tiedostoon.

5.4 Käyttöliitymän upload-napin toiminta

CreateUiUploadButton-funktio luo käyttöliittymään button-elementin, joka on sidottu UiUploadButton_Click-funktioon. Tämän kokonaisuuden tarkoituksena on lähettää editoitu tiedosto temp-kansiosta takaisin alkuperäiseen polkuunsa. Funktio käyttää hyväkseen FileHandler-luokan uploadFile-funktiota, joka kopioi tiedoston InitLoader-muuttujalta saamaansa polkuun.

Kun upload-nappia painetaan, tekee ohjelma kyselyn siitä, onko käyttäjä aivan varmasti halukas lähettämään tiedostot. Mikäli käyttäjä painaa ”close”, ohjelma palautuu takaisin ja jos ”ok” niin suoritetaan tiedostojen lähetys.

5.5 Käyttöliittymän tekstikentän ja dynaamisesti määräytyvien osien toiminta

CreateTextBox-funktio. Tätä funktiota käytetään TextBox-luokkaolion UiTextBox alustamiseen. Funktio sisältää esimerkiksi koon ja sijainnin sekä muita toimintoja.

CreateLabel-funktion tarkoitus on olla osa kolmen eri funktion sarjaa, jossa dynaamisesti luodaan textbox, label sekä radiobutton, joidenka sisältö on ohjelmiston FilePath.ini-tiedostosta on parsittu. Tämä funktio saa arvoikseen yhden bool-tyypisen muuttujan sekä yhden string-tyyppisen muuttujan sekä kaksi int-tyyppistä muuttujaa, jotka toimivat koordinaatteina. Tässä funktiossa luodaan aluksi Label-luokan muuttuja CheckLabel, jolle annetaan arvoiksi sen koko, sijainti, nimi käyttöliittymässä näkyvä teksti sekä fontin tyyppi.

Koko tässä tapauksessa on leveys 104 pikseliä sekä korkeus 16 pikseliä. Sijainti tulee koordinaattimuuttujista. Nimi tuli string-tyypin muuttujasta sekä siihen lisätystä ”_infoTextbox”-tekstistä, jolloin sitä voitiin helpommin kutsua listasta haluttaessa myöhemmin. Fontiksi asetettiin DefaultFont ja tyyliksi Bold. Funktiossa on if/else-lause, jonka tulos määräytyy saadusta bool -muuttujasta. Mikäli tulos on ”True”, värjätään fontti vihreäksi ja laitetaan käyttöliittymässä näkyväksi tekstiksi ”SUCCESS” ja mikäli tulos on ”False”, värjätään fontti punaiseksi ja laitetaan tekstiksi ”FAIL”.

CreateNameBox-funktio on myös osa yllä mainittua sarjaa. Tämä funktio saa arvoikseen yhden string-tyypin muuttujan sekä kaksi int-tyypin muuttujaa. Tässä funktiossa luodaan aluksi TextBox-luokan muuttuja NameBox, jolle annetaan arvoiksi koko, sijainti, nimi, käyttöliittymässä näkyvä teksti sekä fontin tyyppi. Koko tässä tapauksessa on leveys 104 pikseliä sekä korkeus 16 pikseliä. Sijainti tulee koordinaattimuuttujista. Nimi tuli string-tyypin muuttujasta sekä siihen lisätystä ”_CheckButton”-tekstistä, jolloin sitä voitiin helpommin kutsua listasta haluttaessa myöhemmin.

Fontiksi asetettiin DefaultFont ja tyyliksi Regular. Käyttöliittymässä näkyvä teksti tulee funktiolle string-muuttuja, joka on FilePath.ini-tiedostosta parsittu.

CreateCheckButton on funktio, joka saa yhden string-tyypin muuttujan sekä kaksi int-tyypin muuttujaa, jotka toimivat koordinaatteina. Tämä funktio on viimeinen osa yllä mainittua sarjaa ja sen tehtävä on tuottaa dynaamisesti radiobuttoneita, joilla tiedostojen valinnat hoidetaan. Funktio on sidottu eventti-funktioon radioButton_CheckedChanged, jolla ladataan tiedosto käyttöliittymän tekstikenttään, joka luotiin CreateTextBox-funktiossa. Tiedoston avaamiseen käytetään .Net-Framework-luokkaa StreamReader sekä File.OpenText-toimintoja. Avattavan tiedoston polku saadaan InitLoader-luokan muuttujasta, johon polut on tallennettu. radioButton_CheckedChanged-funktiossa on myös try/catch-tarkistus, jolla yritetään estää ohjelmiston kaatuminen tilanteessa, jossa tiedostoa ei löydy.

5.6 Käyttäjän informointifunktioiden toiminta

ValidateUserUpdate-funktio tuottaa kyselyn upload-nappia painaessa. Kyseessä on MessageBox-luokka, jossa on tekstinä kysymys käyttäjälle siitä, haluaako käyttäjä varmasti lähettää tiedostot ja kaksi näppäintä: kyllä ja peruuta. Funktio palauttaa joko ”true” tai ”false”.

InformationPopup on funktio, joka saa arvokseen string-tyyppisen muuttujan. Tämä messagebox on tarkoitettu informaation lähettämiseen käyttäjälle esimerkiksi save-näppäintä painaessa, jolloin kuvaruutuun avautuu messagebox, jossa kerrotaan käyttäjälle, että tämä tiedosto on nyt tallennettu. Boksissa on ”ok”-näppäin, jota painamalla boksi sulkeutuu.

CreateUiHeaderLabel-funktio, joka saa arvoikseen yhden string-tyypin muuttujan sekä kaksi int-tyypin muuttujaa, jotka ovat koordinaatteja. Tämä funktio lisää tekstin radiobuttoneiden väliin, jolla eritellään editoitavat tiedostot sekä tarkistettavat ohjelmistot toisistaan. Näin tehtynä käyttöliittymä pysyy selkeämpänä ja käyttäjä erehtyy harvemmin.

6 Jatkokehitys

6.1 Nykyinen tilanne

Tuloksena saatiin ohjelmisto, jota tullaan lisäkehittämään. Ohjelmistossa on sen alkuun halutut ominaisuudet, mutta koska työn vaiheessa huomattiin, että tarvitaan lisäominaisuuksia, niin niiden suunnittelu on vielä kesken. Nyt tehtyihin tuloksiin päästiin omalla suunnittelutyöllä sekä käyttämällä saatavilla olleita resursseja sekä tietoja erinäisistä lähteistä. Työn onnistuminen oli kohtalaista, sillä ominaisuudet, jotka alkuperäisesti haluttiin, on tehty, mutta ohjelman toiminnassa on vielä puutteita, esimerkiksi kaikista mahdollisista virheistä ei tule vielä oikean muotoista virheilmoitusta ohjelman lokiin.

Toimeksiantaja on ilmaissut tyytyväisyytensä ohjelmistoon ja haluaa sitä lisäkehitettävän. Toimeksiantaja on myös ilmaissut ymmärryksensä sille, että ohjelmistoon tulleiden muutosten takia kaikkia ominaisuuksia ei vielä ole implementoitu.

Työssä olisi pitänyt sen vaativuuden takia käyttää enemmän aikaa ja resursseja suunnitteluun, jotta ohjelmiston lähdekoodi olisi rakenteeltaan selkeämpi sekä luettavampi. Ohjelmistossa olisi myös heti alussa pitänyt ottaa tarkempaan tarkasteluun vaatimusmäärittelyt, joita nyt tuli kesken työn implementoitavaksi.

Työssä opittiin ainakin se, että hyvin suunniteltu on puoliksi tehty. Työn aloitusvaiheessa tulisi käyttää ohjelmiston arkkitehtuurin suunnitteluun paljon enemmän aikaa sekä vaatimusmäärittelyjen laatimiseen täsmällisyyttä. Työssä opittiin myös paljon ohjelmointitekniikoita, jotka olivat ennestään tuntemattomia tekijälle. Työ myös tutustutti tekijän palvelinympäristöihin sekä ohjelmistoihin, joista ei ollut aikaisemmin kokemusta.

Tämän työn aiheesta ja sen tarpeesta voidaan tehdä ainakin se johtopäätös, että ohjelmistoja suunnitellessa tulisi aina kiinnittää huomiota myös sen konfigurointiin asennusvaiheessa sekä asetusvälilehtien suunnitteluun. Tällöin säästyttäisiin ongelmilta ohjelmiston elinkaarta ajatellen, koska monesti ne, jotka ovat ohjelmiston aikoinaan suunnitelleet ovat jo jatkaneet uraansa jossain muulla eikä välttämättä yritystäkään, joka ohjelmiston on luonut, ole enää olemassa.

6.2 Tilanne tulevaisuudessa

Asiakasyritys on ilmaissut halunsa jatkaa projektia ja sen kehitystä. Ohjelmistoon on mietitty lisättäväksi muun muassa asiakasohjelmiston lokien luku, jolla saataisiin vian etsintää nopeutettua. Tällä hetkellä asiakasohjelmiston lokit sekä siihen liitoksissa olevien muiden ohjelmistojen lokit ovat erinäisten kansio puiden alla ja niiden löytämiseen sekä avaamiseen menee kohtuuttomasti aikaa. Esimerkiksi hiemankaan hankalamman vian löytämiseen saatetaan joutua avaamaan ja tarkastamaan kolmesta neljään eri lokitiedostoa eri paikoista, joten tämän takia olisi ne hyvä olla kaikki kerättynä samaan paikkaan konfiguraatio-ohjelmiston avulla. Konfiguraatio-ohjelmisto myös taipuisi tähän helposti sen suurempaa ohjelmointia vaatimatta, sillä kaikki nämä ominaisuudet ovat jo olemassa ja ohjelmiston dynaamisen arkkitehtuurin takia voidaan polut vain kiinnittävät FilePath.ini-tiedostoon, jolloin edes ohjelmistoa ei tarvitse uudelleen kääntää.

On myös muita mahdollisia jatkokehitystoimenpiteitä pohdittu esimerkiksi, että konfiguraatio-ohjelmisto voisi automaattisesti tarkastaa ja muuttaa jo ennalta tiedetyt konfigurointitiedostot, sekä tarkastaa Windowsin rekisteristä lokalisaatioasetukset, kuten järjestelmän kielen, näppäimistön kielen sekä pilkun ja pisteen käytön erottimissa. Nämä kaikki tiedot ovat aika helposti muokattavissa Windows-rekisteristä ja .Net-frameworkissa on myös valmiit luokat ja funktiot rekisterin editointia varten.

Envisetin käyttöliittymää voisi myös muuttaa, jolloin se voisi parsia konfiguraatiotiedostoista kentät niin, että vain ne kohdat, joita voi ja saa muuttaa olisivat tekstikenttiä ja muu teksti olisi suojattu muutokselta.

On myös pohdittu mahdollisuutta lisätä uutena ominaisuutena SQL- ja Oracle-tietokantojen eheyden tarkistus sekä erinäisiä valmiita komentoja Oracle-ympäristöä ajatellen. Oracle tarjoaa täyden ohjattavuuden ja tietokantojen tarkastelun suoraan komentokehotteesta, mutta nämä komennot vaativat kohtalaisen korkeaa osaamista ja ymmärrystä, jotenka olisi hyvä saada jokin ohjelmisto kevyeen Oraclen tarkasteluun ja ohjaamiseen. Tässä olisi Enviset-ohjelmistolle hyvin mahdollisuuksia tehdä jonkinlainen käyttöliittymä esimerkiksi ennalta määrättyihin SQL-kyselyihin, joilla voitaisiin tarkastaa tietueiden eheyttä tai Oraclen-moottorin virheilmoituksia tarkastelua varten.

7 Pohdinta

Tämä työ oli opettavainen monella eri tavalla. Tekijän suunnitteluosaaminen on kehittynyt työn edetessä huomattavasti.

Työn jälkeen huomattiin, että projektia olisi pitänyt ohjata aivan toisin. Yhdeksänkymmentäluvun Postit-lapputekniikka ja varmuuskopiointi manuaalisella kansiorakenteella ei ole tätä päivää. Myös työn suunnitteluun ja ohjelmiston arkkitehtuurissa olisi pitänyt käyttää useampia henkilöitä. Sekä asiakasyrityksen että asiakasyrityksen asiakkaan kanssa olisi pitänyt pystyä keskustelemaan asiasta palavereissa enemmän ja tiiviimmin, mutta tähän ei aikaresurssit riittäneet kenenkään puolelta. Tarjolla olisi ollut tikettijärjestelmä ja dokumentointijärjestelmä, mutta aikataulupaineiden vuoksi kyseiset järjestelmät eivät olleet käytettävissä. Ohjelmiston sisäinen rakenne oli suunniteltu alkuperäisiä tarpeita varten, mutta koska ympäristö, jossa ohjelmiston tuli toimia, vaihtui ja sen vaatimukset myös muuttuivat kesken työn, on sen toimintaa tarkasteltava vielä uudestaan. Arkkitehtuuriohjelmistossa on tekijän kehittämä ja koska tekijä ei ollut aikaisemmin samantyyppisiä järjestelmiä kehittänyt, se näkyy osittain laadussa. Kaiken kaikkiaan tekijän alussa oleva ura sekä uusi työympäristö aiheuttivat sen, että jatkokehityksessä laadun tarkastelu on avainasemassa. Tämän työn suurin anti oli se, että työn tekijän osaaminen kyllä karttui seuraavia projekteja varten ja myös työympäristö sekä asiakasyritys ja sen toimintatavat niin hyvässä kuin pahassa tulivat tutuiksi.

Ohjelmistosuunnittelun työtavat sekä toimintamallit alkoivat hahmottua työn loppuvaiheissa sekä lisätieto ohjelmointityylistä saatiin viime hetkellä, jolloin vielä joitakin asioita koodissa saatiin muokattua, esimerkiksi kommenttikentät lisättiin koodiin oikeaan muotoonsa, mutta esimerkiksi funktioiden nimet sekä muuttujien nimiin ei haluttu enää jälkikäteen koskea, jotta ei rikottaisi mitään. Myös työkalujen käytöstä kuten Visual Studiosta, opittiin uutta, mutta sen tehokäyttöön ei vielä päästy. Kaiken kaikkiaan projektissa oli haasteita toteuttaa ja sovittaa yhteen asiakasyrityksen sisäisten sääntöjen kanssa.

Lähteet

1 Gregory, J. Game Engine Architecture. 3d ed. Boca Raton: Taylor & Francis, CRC Press ; 2018 [viitattu 18.8.2023].

2 Wikipedia: Coding best practices [Internet] [viitattu 23.9.2023]. Saatavilla: https://en.wikipedia.org/wiki/Coding_best_practices

3 Wikipedia: Maintainability [Internet] [viitattu 23.9.2023]. Saatavilla: https://en.wikipedia.org/wiki/Maintainability

4 Wikipedia: Dependability [Internet] [viitattu 23.9.2023]. Saatavilla: https://en.wikipedia.org/wiki/Dependability

5 Wikipedia: Usability [Internet] [viitattu 23.9.2023]. Saatavilla: https://en.wikipedia.org/wiki/Usability

6 Microsoft. Microsoft Learn. [Internet] [viitattu 23.9.2023] Saatavilla: https://learn.microsoft.com/en-us/dotnet/csharp/

7 Wim Hoogenraad. Mikä on manuaalinen testaus? [Internet] [viitattu 13.5.2023]. Saatavilla: https://fi.itpedia.nl/2017/10/11/wat-is-handmatig-testen/

8 Marko Wegelius. Käyttöliittymä kuva [viitattu 7.9.2023]