

Saimaan ammattikorkeakoulu  
Tekniikka Lappeenranta  
Tietotekniikan koulutusohjelma  
Tietojärjestelmien kehitys

Simo Nenonen & Visa Pessa

## **Katsastusaseman työvuorojen suunnitteluohjelma**

Opinnäytetyö 2014

## Tiivistelmä

Simo Nenonen & Visa Pessa

Katsastusaseman työvuorojen suunnitteluohjelma, 65 sivua

Saimaan ammattikorkeakoulu

Tekniikka Lappeenranta

Tietotekniikan koulutusohjelma

Tietojärjestelmien kehitys

Opinnäytetyö 2014

Ohjaajat: lehtori Martti Ylä-Jussila, Saimaan ammattikorkeakoulu,

katsastusinsinööri Ari Kultanen, A-Katsastus Oy

aluepäällikkö Jari Joenperä, A-Katsastus Oy

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa määritelty työvuorojen suunnitteluohjelma A-Katsastus Oy:lle. Ohjelmalla voidaan luoda automaattisesti työvuorolistat yrityksen työntekijöille, jonka jälkeen listoja voidaan tarpeen vaatiessa muokata.

Ohjelman määrittelyn oli tehnyt Ari Kultanen omassa opinnäytetyössään, joten tehtävänä oli suunnitella ja toteuttaa ohjelma. Työssä käytettiin C#-ohjelmointikieltä, koska näin oli päätetty määrittelyssä, sekä MySQL-tietokantaa Accessin sijasta, jota määrittelyssä oli ehdotettu.

Opinnäytetyön tuloksena asiakkaalle tuotettiin toimiva ja vaatimusten mukainen katsastusaseman työvuorojen suunnitteluohjelma sekä sen käyttöohje.

Asiasanat: C#, ohjelmistotestaus, ohjelmointi, tietokanta, Visual Studio, työvuorojen suunnittelu

## **Abstract**

Simo Nenonen & Visa Pessa

Rota designing program for vehicle inspection station, 65 pages

Saimaa University of Applied Sciences

Technology Lappeenranta

Information Technology Degree Programme

Development of Information Systems

Bachelor's Thesis 2014

Instructors: lecturer Martti Ylä-Jussila, Saimaa University of Applied Sciences,  
inspection engineer Ari Kultanen, A-Katsastus Ltd.  
regional manager Jari Joenperä, A-Katsastus Ltd.

The purpose of this thesis was to study programming and to design and implement a planning program for rotas for A-Katsastus Ltd. The program can be used to create rotas for employees. The rotas can be modified afterwards.

The requirement specification was done by Ari Kultanen in his own thesis so designing and implementing the program itself was the job. C#-programming language was used because that was decided in the requirement specification. MySQL-database was chosen for implementing the program over Access that was the database of choice in the requirement specification.

The result of this thesis is a working rota designing program for vehicle inspection station that meets the requirements that were set to it and also a manual.

Keywords: C#, software testing, programming, database, Visual Studio, rota design

## Sisällysluettelo

Käsitteet .....	6
1 Johdanto .....	8
2 A-Katsastus .....	9
2.1 Autokatsastus .....	9
2.2 Työvuorojen jako .....	10
3 Ohjelmistotuotanto .....	12
3.1 Vaihejakomallit .....	13
3.1.1 Vesiputousmalli .....	14
3.1.2 Prototyyppimenetelmä .....	14
3.1.3 Rational Unified Process, RUP .....	15
3.2 Ohjelmiston elinkaari ja vaiheet .....	16
3.2.1 Vaatimusanalyysi .....	16
3.2.2 Järjestelmäsuunnittelu .....	16
3.2.3 Ohjelmistosuunnittelu .....	17
3.2.4 Toteutus .....	18
3.2.5 Testaus .....	18
3.2.6 Julkaisu/käyttöönotto .....	18
3.2.7 Ylläpito .....	19
3.2.8 Elinkaaren loppu .....	19
3.3 Ketterä ohjelmistokehitys .....	19
3.3.1 Scrum .....	19
3.3.2 Extreme programming, XP .....	21
4 Ohjelmiston testaaminen .....	26
4.1 Testausmetodit .....	27
4.2 Testaustasot .....	28
4.3 Testaustyytit .....	29
4.4 Testaustyökalut .....	31
5 Arkkitehtuuri .....	31
5.1 Ohjelmistoarkkitehtuuri .....	31
5.2 Suunnittelumallit .....	33
5.2.1 Antisuunnittelumallit .....	34
5.2.2 Hyödyt ja haitat .....	34
5.3 Laitteistoarkkitehtuuri .....	35
5.4 Ajonaikainen arkkitehtuuri .....	35
5.4.1 Ajonaikaisen arkkitehtuurin kuvaaminen .....	35
6 Käytetyt tekniikat .....	37
6.1 Microsoft .NET .....	37
6.1.1 Ominaisuudet .....	37
6.1.2 Arkkitehtuuri .....	38
6.2 C#-ohjelmointikieli .....	39
6.3 Relaatiotietokanta .....	40
6.3.1 Microsoft Access .....	40
6.3.2 MySQL .....	40
6.3.3 Tietokantayhteydet .....	41
6.4 SQL-kieli .....	42
6.4.1 Esimerkki tietokantakyselyn käytöstä .....	43
7 Työvälineet .....	44

7.1	Microsoft Visual Studio 2010 .....	44
7.1.1	Koodieditori .....	45
7.1.2	IntelliSense.....	45
7.1.3	Testaustyökalu .....	46
7.2	XAMPP .....	47
8	Opinnäytetyöprojektin toteutus .....	48
8.1	Projektin suunnitteluvaiheet ja aikataulu .....	48
8.1.1	Opiskelu .....	48
8.1.2	Suunnittelu ja ohjelmointi .....	49
8.1.3	Ohjelman testaaminen ja korjaaminen .....	50
8.2	Tehtävien työmäärät .....	50
9	Lopputuotteen esittely .....	51
9.1	Ohjelman tietokantarakenne .....	51
9.2	Sisäänkirjautuminen .....	52
9.3	Valikko .....	52
9.4	Työvuorojen hallinta.....	53
9.5	Työntekijöiden hallinta .....	56
9.6	Merkintätyyppien hallinta .....	57
9.7	Toimipisteiden hallinta .....	59
9.8	Työntekijän kalenterimerkintä .....	60
9.9	Yleistä .....	60
10	Johtopäätökset.....	61
	Kuvat.....	63
	Lähteet.....	64

## Käsitteet

.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto.
Access	Access on Microsoftin kehittämä relaatiotietokantaohjelma.
Apache	Apache on avoimeen lähdekoodin perustava HTTP-palvelin ohjelma.
C#	C# on Microsoftin kehittämä oliopohjainen ohjelmointikieli.
Entity Framework	Entity Framework on ORM-kehys.
Google Drive	Google Drive on Googlen tarjoama pilvipalvelu.
IntelliSense	IntelliSense on Microsoftin kehittämä koodin täydennystyökalu, joka on tietoinen ympäristöstään.
KATASO	Katsastusaseman työvuorojen suunnitteluohjelma.
MySQL	MySQL on relaatiotietokantaohjelmisto.
ORM	Object Relational Mapping.
Perl	Perl on ohjelmointikieli ja se on erityisen suosittu WWW-ohjelmoijien keskuudessa.
PHP	PHP eli Hypertext Preprocessor on Perlin kaltainen ohjelmointikieli. Sitä voidaan käyttää useilla eri alustoilla ja käyttöjärjestelmillä.
RUP	Rational Unified Process on interatiivisen ohjelmistokehityksen prosessikehys.
Scrum	Ketterä ohjelmistokehitysmenetelmä.
SQL	SQL eli Structured Query Language on kyselykieli, jolla relaatiotietokantaan voi tehdä erilaisia hakuja.
Transaktio	Yhtenä toimenpiteenä suoritettu hakujen ja/tai tallennusten sarja, joka voidaan tarpeen tullessa peruuttaa.
UML	Unified Modeling Language eli standardisoitu graafinen mallinnuskieli. Sillä tehdään muun muassa käyttäytymis-, vuorovaikutus- ja rakennekaavioita.
Visual Studio	Microsoftin ohjelman kehitysympäristö, jossa voidaan käyttää useita eri ohjelmointikieliä kuten C++ ja C#.

Windows Forms

Graafinen ohjelmointirajapinta.

XAMPP

X = Cross Platform, A = Apache HTTP Server, M = MySQL, P = PHP, P = Perl. XAMPP on avoimen lähdekoodin ohjelma, joka on tarkoitettu apuvälineeksi sivustojen ja ohjelmien suunnitteluun.

XP

Extreme programming on ketterä ohjelmistokehitysmenetelmä.

# **1 Johdanto**

Opinnäytetyön tarkoituksena on suunnitella ja toteuttaa A-Katsastukselle määritelly työvuorojen suunnitteluohjelma. Määrittelyn on tehnyt opinnäytetyönään Ari Kultanen.

Perimmäinen ongelma, josta projekti lähti liikkeelle, on työvuorojen tämänhetkisen suunnittelun tehottomuus. Työvuorolistat tehdään käsin Exceliä käyttäen, mikä on hidasta ja virheitä sattuu helposti monimutkaisen vuorojen jaon takia. Valmiin työvuorojen suunnitteluohjelman ostaminen oli tässä tapauksessa riski, koska niiden yhteensopivuus katsastusaseman monimutkaisen työvuorojärjestelmän kanssa oli huono. Näin päädyttiin tekemään ohjelma itse.

Työvuorojen suunnitteluohjelma tulee käyttöön pilottiversiona vain A-Katsastuksen Lappeenrannan yksikköön yhdelle toimipisteelle. Ohjelma saattaa levitä myös muihin toimipisteisiin.



## **2 A-Katsastus**

A-Katsastus on osa A-Katsastus Group -konsernia, joka on johtava yksityinen ajoneuvokatsastusten, rekisteröintien, kuljettajatutkintojen ja ajoneuvontestauspalveluiden tarjoaja Pohjois-Euroopassa. A-Katsastus toimii pääosin Suomessa mutta on levinnyt myös Latviaan, Puolaan, Ruotsiin, Tanskaan ja Viroon. Konsernissa on noin 300 katsastusasemaa ja sen pääkonttori sijaitsee Helsingissä. A-Katsastuksen historia alkaa jo 1900-luvun alusta, jolloin ensimmäiset ajoneuvojen katsastukset ja kuljettajatutkinnot suoritettiin Suomessa. (A-Katsastus Oy 2013.)

### **2.1 Autokatsastus**

Auton katsastuksesta määrätään valtioneuvoston asetuksessa kuvan mukaisesti (Kuva 1).

Ajoneuvoluokka	Määräaikauskatsastus on suoritettava
a) Linja- ja kuorma-autot (M <sub>2</sub> -, M <sub>3</sub> -, N <sub>2</sub> - ja N <sub>3</sub> -luokka), erikoisautot, joiden kokonaismassa on suurempi kuin 3,5 tonnia, luvanvaraiseen liikenteeseen käytettävät henkilöautot (M <sub>1</sub> -luokka) sekä sairausautot	Ensimmäisen kerran viimeistään vuoden kuluttua ajoneuvon käyttöönottopäivästä ja sen jälkeen vuosittain viimeistään käyttöönottopäivää vastaavana päivänä
b) Perävaunut, joiden kokonaismassa on suurempi kuin 3,5 tonnia (O <sub>3</sub> - ja O <sub>4</sub> -luokka)	Ensimmäisen kerran viimeistään vuoden kuluttua ajoneuvon käyttöönottopäivästä ja sen jälkeen vuosittain viimeistään käyttöönottopäivää vastaavana päivänä; kytkentäkatkautuksessa tiettyyn vetoautoon kytketty perävaunu saadaan kuitenkin tuoda määräaikauskatsastukseen yhtä aikaa vetoauton kanssa
c) Pakettiautot (N <sub>1</sub> -luokka) ja sairausautoja lukuun ottamatta erikoisautot, joiden kokonaismassa on enintään 3,5 tonnia	Ensimmäisen kerran kolmen vuoden kuluttua ajoneuvon käyttöönottopäivästä ja sen jälkeen vuosittain viimeistään käyttöönottopäivää vastaavana päivänä
d) Yksityiseen liikenteeseen käytettävät henkilöautot ja muut M1-luokan ajoneuvot kuin sairausautot, kevyet nelipyörät (L <sub>6e</sub> -luokka) sekä nelipyörät (L <sub>7e</sub> -luokka)	Ensimmäisen kerran kolmen vuoden kuluttua ajoneuvon käyttöönottopäivästä, toisen kerran viiden vuoden kuluttua ajoneuvon käyttöönottopäivästä ja sen jälkeen vuosittain viimeistään käyttöönottopäivää vastaavana päivänä
e) Perävaunut, joiden kokonaismassa on suurempi kuin 0,75 tonnia mutta enintään 3,5 tonnia (O <sub>2</sub> -luokka)	Ensimmäisen kerran kalenterivuoden loppuun mennessä sinä vuonna, jolloin käyttöönottopäivästä on kulunut kaksi vuotta, ja sen jälkeen kahden vuoden välein kalenterivuoden loppuun mennessä
f) 1 päivänä tammikuuta 1960 tai sen jälkeen käyttöön otetut katsastusvelvolliseen ajoneuvoluokkaan kuuluvat museoajoneuvot	Kahden vuoden välein kesäkuun loppuun mennessä
g) Ennen 1 päivää tammikuuta 1960 käyttöön otetut katsastusvelvolliseen ajoneuvoluokkaan kuuluvat museoajoneuvot	Neljän vuoden välein kesäkuun loppuun mennessä

Kuva 1. Valtioneuvoston asetus liikenteessä käytettävien ajoneuvojen liikenne-kelpoisuuden valvonnasta (Edita Publishing Oy, 19.12.2002/1245 –kuva)

Kuva 1 esittää erilaisten autotyyppien katsastusaikatauluja.

Katsastajilla on erilaisia katsastusoikeuksia, jotka on määrätty ajoneuvon katsastusluvista määrittelevässä laissa. Katsastajilla voi myös olla erikoiskatsastusoikeuksia, jotka myös ovat laissa määrättyjä. Niitä ovat kevyen kaluston rekisteröinti- ja muutokatsastus, raskaan kaluston rekisteröinti- ja muutokatsastus sekä paineilmajarrut.

## 2.2 Työvuorojen jako

Työvuorojen jakamisessa pyritään käyttämään tasapuolisuutta, mutta siinä pitää ottaa myös huomioon työntekijöiden erilaiset katsastusoikeudet. Esimiehet täyt-

tävät työvuorot Excel-taulukkoon nykyisessä järjestelmässä. Näin ollen tasa-puolisuus on täysin esimiehin vallassa.

Työpaikan työvuorot tulee jakaa lain mukaisesti. Työaika riippuu katsastuspis-teen aukioloajoista ja tarjottavista palveluista. A-Katsastuksella työaika on 37,5 tuntia viikossa. Työvuorojen jakamisesta säädetään työaikalaissa seuraavasti (Työaikalaki 9.8.1996/605):

#### *4 § Työaika*

*Työajaksi luetaan työhön käytetty aika sekä aika, jonka työntekijä on velvollinen olemaan työpaikalla työnantajan käytettävissä.*

*Jäljempänä 28 §:ssä tarkoitettuja tai sopimukseen perustuvia päivittäisiä lepoai-koja ei lueta työaikaan, jos työntekijä saa näinä aikoina esteettömästi poistua työpaikalta.*

*Matkaan käytettyä aikaa ei lueta työaikaan, ellei sitä samalla ole pidettävä työ-suorituksena.*

#### *6 § Yleissäännös*

*Säännöllinen työaika on enintään kahdeksan tuntia vuorokaudessa ja 40 tuntia viikossa.*

*Viikoittainen säännöllinen työaika voidaan järjestää myös keskimäärin 40 tun-niksi enintään 52 viikon ajanjakson aikana.*

#### *7§ Jaksotyöaika*

*Säännöllinen työaika saadaan 6 §:ssä säädetystä poiketen järjestää niin, että se on kolmen viikon pituisena ajanjaksona enintään 120 tuntia tai kahden viikon pituisena ajanjaksona enintään 80 tuntia.*

*Työn tarkoituksenmukaiseksi järjestämiseksi tai työntekijöille epätarkoituksen-mukaisten työvuorojen välttämiseksi voidaan säännöllinen työaika 1 momentis-sa säädetystä poiketen järjestää niin, että se on kahden toisiaan seuraavan kolmen viikon ajanjakson aikana tai kolmen toisiaan seuraavan kahden viikon ajanjakson aikana enintään 240 tuntia. Säännöllinen työaika ei saa kumman-kaan kolmen viikon ajanjakson aikana ylittää 128 tuntia eikä yhdenkään kahden viikon ajanjakson aikana 88 tuntia.*

#### *13 § liukuva työaika*

*Työnantaja ja työntekijä voivat 6 §:n 1 momentista sekä työehtosopimuksen säännöllisen työajan pituutta ja sijoittamista koskevista määräyksistä poiketen sopia liukuvasta työajasta niin, että työntekijä voi sovituissa rajoissa määrätä työnsä päivittäisen alkamis- ja päättymisajankohdan. Sovittaessa liukuvasta työajasta on sovittava ainakin kiinteästä työajasta, työajan vuorokautisesta liu-*

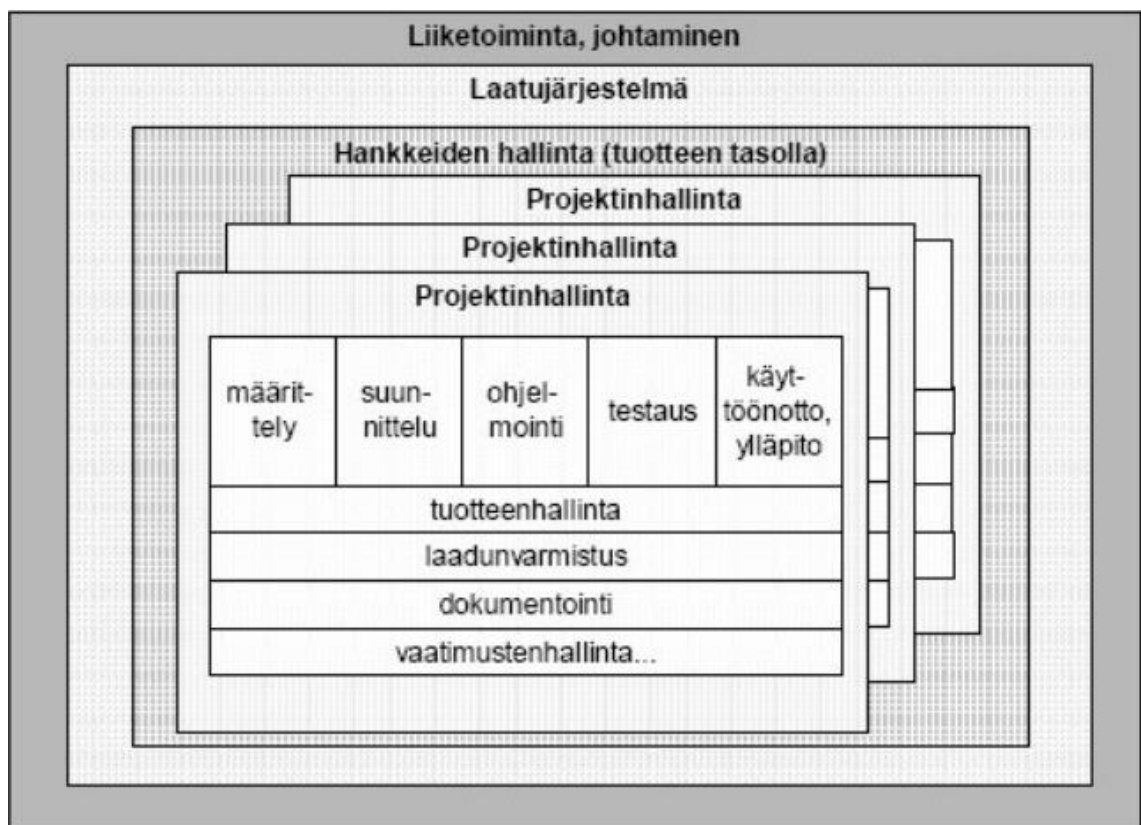
kumarajasta, lepoaikojen sijoittamisesta sekä säännöllisen työajan ylitysten ja alitusten enimmäiskertymästä.

Liukuvassa työajassa säännöllistä vuorokautista työaikaa lyhentää tai pidentää liukuma-aika, joka voi olla enintään kolme tuntia. Viikoittainen säännöllinen työaika on keskimäärin enintään 40 tuntia. Edellä 1 momentissa mainittu enimmäiskertymä saa olla enintään 40 tuntia.

Työnantaja ja työntekijä voivat sopia, että työajan ylitysten kertymää vähennetään työntekijälle annettavalla vapaa-ajalla.

### 3 Ohjelmistotuotanto

Ohjelmistotuotanto liiketoimintana tuli ensimmäisten kaupallisten tietokoneiden myötä 1950-luvulla. Kuva 2 esittää ohjelmistotuotannon osa-alueita.



Kuva 2. Ohjelmistotuotannon osa-alueet (Haikala & Märijärvi 2004, s. 35.)

Ohjelmistotuotantoon (Kuva 2) sisältyy kaikki työnteon ja työnjohdon menetelmät, joita käytetään tietokoneohjelmien/-ohjelmistojen tuotannossa.

Tuotteenhallinta on versionhallintaa, nimeämiskäytäntöjä, konfiguraation sovel-  
lusmenetelmiä, muutosten hallintaa sekä arkistointia. (Haikala & Märijärvi 2004,  
s. 35–60.)

Laadunvarmistuksella tarkoitetaan virheiden estämistä sekä niiden etsimistä.  
Tätä voidaan suorittaa erilaisilla laatujärjestelmillä. (Haikala & Märijärvi 2004, s.  
35–60.)

Dokumentointi on tärkeää tuotteen jatkon kannalta. Dokumentointiin kuuluu vä-  
hintään projektisuunnitelma, määrittelydokumentti, suunnitteludokumentti ja tes-  
taussuunnitelma. (Haikala & Märijärvi 2004, s. 35–60.)

Vaativuushallinnalla tarkoitetaan, että lopputuote vastaa asiakkaan vaati-  
muksia. (Haikala & Märijärvi 2004, s. 35–60.)

Projektinhallinta sisältää projektin sisäisten resurssien hallinnointia sellaisella  
tavalla, että projektille asetetuissa rajoissa pysytään, esimerkiksi budjetin tai ai-  
kataulun kannalta. (Haikala & Märijärvi 2004, s. 35–60.)

Hankkeiden hallinta on hankekokonaisuuden jäsentämistä, jotta hanke toteutuu  
suunnitellusti. (Haikala & Märijärvi 2004, s. 35–60.)

Laatujärjestelmä kuvaa laadunhallinnan ja laatu toiminnan kokonaisuutta. (Hai-  
kala & Märijärvi 2004, s. 35–60.)

Liiketoiminta käsittää yrityksen liiketoiminnan, eli millä alueella ja miten liiketoi-  
mintaa suoritetaan. (Haikala & Märijärvi 2004, s. 35–60.)

Johtaminen tarkoittaa yrityksen johtamistapoja, joita ovat suhtautuminen henki-  
löstöön, henkilöasioiden organisointi, henkilöstöresurssien käyttöä sekä työelä-  
män suhteiden hoitamista. (Haikala & Märijärvi 2004, s. 35–60.)

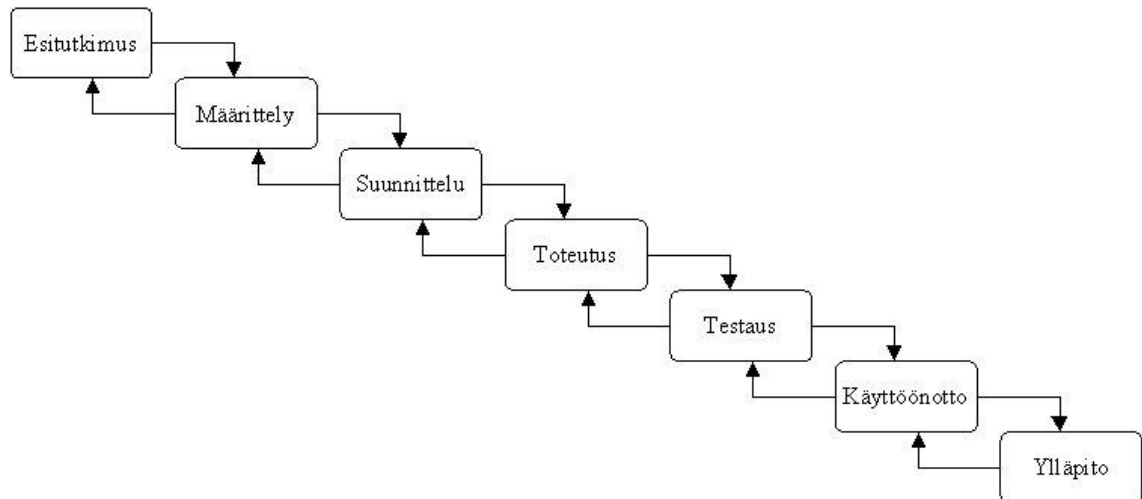
### **3.1 Vaihejakomallit**

Ohjelmistotuotantoon on erilaisia vaihejakomalleja, joista kaikki ovat tehokkaita  
tapoja hallita ohjelmistokehitystä. Ohjelmiston kehitys on monimutkainen tehtä-  
vä, jonka avuksi tarvitaan projektinhallinta- ja suunnittelumenetelmiä. Teollisuus-

dessa yleisesti käytettäviä projektinhallintamalleja ovat vesiputousmalli, prototyypimenetelmä ja RUP-malli. (Haikala & Mikkonen 2011, s. 36.)

### 3.1.1 Vesiputousmalli

Vesiputousmalli nimensä mukaisesti valuu alaspäin, mutta oikeasti siinä palataan myös tarvittaessa taaksepäin korjaamaan vaadittuja asioita. Vesiputousmallin vaiheet on nähtävissä kuvassa Kuva 3.



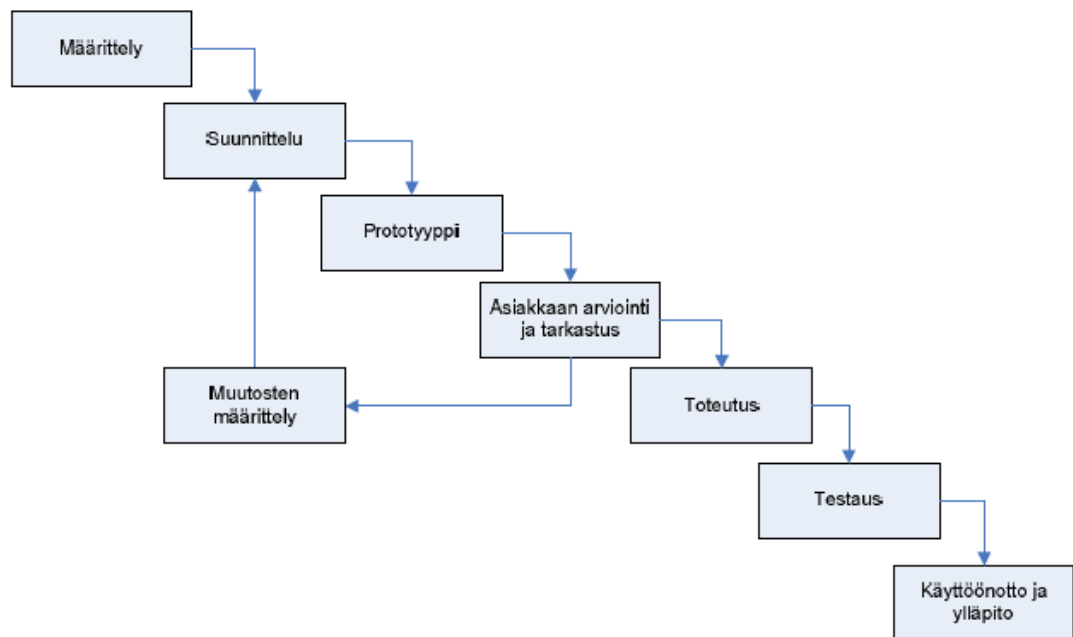
Kuva 3. Vesiputousmalli (Oulun seudun ammattiopisto. Vesiputousmalli-kuva)

Kuva 3 näyttää vesiputousmallin kaikki vaiheet alusta loppuun. Edellisen vaiheen on tarkoitus tuottaa tarvittava dokumentaatio, jotta seuraava vaihe voidaan toteuttaa. Teoreettinen ajatus on vesiputousmallissa hyvä, mutta käytännön toteutus mallia noudattaen on hankalaa. Usein ohjelmistokehityksessä taustalta löytyy vesiputousmallia muistuttava rakenne, mutta sitä ei kuitenkaan noudateta orjallisesti. Käytännössä kehitys on iteratiivista, jolloin ohjelman toiminnot kasvavat pikkuhiljaa. Vesiputousmalli on hyvä malli kaikille ohjelmistokehitysprojekteille. (Haikala & Mikkonen 2011, s. 36–38.)

### 3.1.2 Prototyypimenetelmä

Prototyypimenetelmä on spiraalimaisin muodoin etenevä ohjelmistokehitysmenetelmä. Se aloitetaan käyttöliittymästä, jolloin asiakas näkee heti, millainen ohjelmasta tulee, ja sen jälkeen vasta kehitetään ohjelmointilogiikat sekä tietokannat. Tällainen etenemistyyli mahdollistaa asiakkaan palautteen tuotteen ulkonäöstä heti alussa. Prototyypimenetelmä toimii hyvin myös tapauksissa, jos-

sa asiakkaan vaatimukset muuttuvat usein. Kuva 4 esittää prototyypin menetelmän etenemistä. (Haikala & Mikkonen 2011, s. 38–39.)

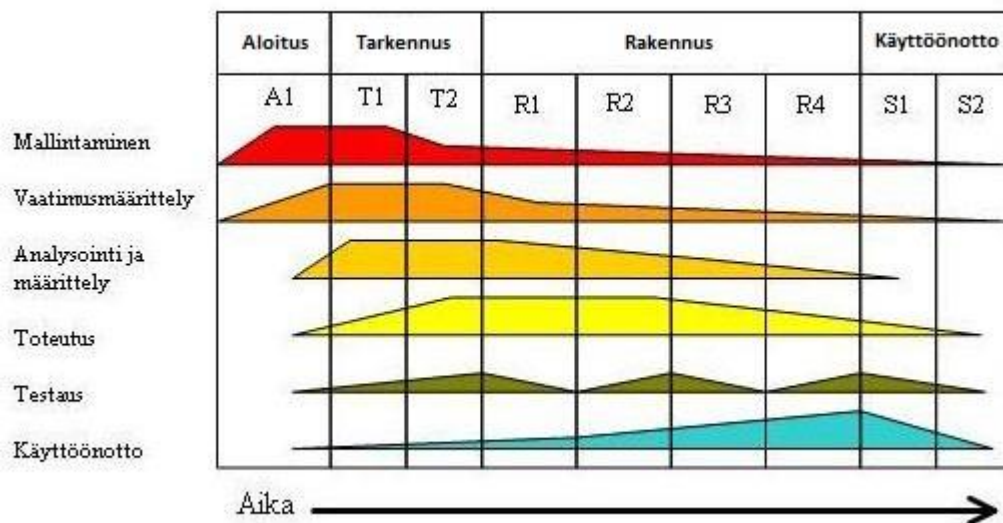


Kuva 4. Protoilu (Kettunen 2009, s.13)

Ns. "protoilu" (Kuva 4) etenee iteraatioissa ja jokaisen kierroksen jälkeen tuotetta esitellään asiakkaalle. Ohjelma on valmis, kun asiakas on hyväksynyt sen.

### 3.1.3 Rational Unified Process, RUP

RUP on todella laaja ohjelmistokehityksen prosessikehikko. Kuva 5 esittää RUP:n toimintoja ja vaiheita.



Kuva 5. RUP (Wikipedia. RUP-kuva)

RUP jakautuu neljään vaiheeseen: aloitus, tarkennus, rakennus ja käyttöönotto. Jokaisessa vaiheessa on monia iterointeja, joista jokaisessa hyödynnetään vesiputousmallia (Kuva 5). RUP on hyvä ja laaja pohja, josta kehittää oma projektin työskentelytapa. (IBM. RUP.)

### 3.2 Ohjelmiston elinkaari ja vaiheet

Ohjelmiston elinkaareissa on kaksi päävaihetta: kehitys ja ylläpito, joista kehitys jakaantuu useisiin alivaiheisiin. Näitä ovat vaatimusmäärittely, toiminnallinen määrittely, arkkitehtuurisuunnittelu, toteutus, testaus ja julkistus/käyttöönotto. (Haikala & Märijärvi 2004, s. 38–48.)

#### 3.2.1 Vaatimusanalyysi

Vaatimusanalyysissä selvitetään, mitä vaatimuksia valmiin ohjelman tulisi täyttää eli suunnitellaan, mitä ohjelman on tarkoitus tehdä. Vaatimusanalyysi suoritetaan asiakkaan kanssa, jotta vaatimukset täsmäisivät asiakkaan näkemyksiä ohjelmasta. (Haikala & Märijärvi 2004, s. 38–48.)

#### 3.2.2 Järjestelmäsuunnittelu

Järjestelmäsuunnittelussa mietitään järjestelmän arkkitehtuuria eli mitä laitteistoja tarvitaan ja miten ohjelmisto sijoitetaan prosessoreille. Järjestelmäsuunnit-



telussa päätetään myös laitteen sisäiset ohjelmistotarpeet. (Haikala & Märijärvi 2004, s. 38–48.)

### 3.2.3 Ohjelmistosuunnittelu

Ohjelmistosuunnittelu jakaantuu kahteen vaiheeseen: toiminnallinen määrittely sekä tekninen määrittely. (Haikala & Märijärvi 2004, s. 80–81.)

Toiminnallisessa määrittelyssä määritetään järjestelmän toiminnalliset ja ei-toiminnalliset vaatimukset. Toiminnallinen määrittely antaa kuvauksen, mitä järjestelmään sisältyy ja miten sen on toimittava:

- järjestelmän ympäristön sekä käyttäjät
- tietokantarakenteen ja tiedot
- järjestelmän toiminnot kuvauksineen
- järjestelmän liittymät sen ulkopuolelle; käyttöliittymä, laitteisto-, ohjelmisto- sekä tietoliikenneliittymät
- ei-toiminnalliset vaatimukset; suorituskky, käytettävyyys ja ylläpidettävyys
- suunnittelurajoitteet.

(Haikala & Märijärvi 2004, s. 80–81.)

Ei-toiminnalliset vaatimukset eli laatuvaatimukset on hyvä määrittää ajoissa. Yksi suosittu laatumalli, joka on peräisin ISO/IEC 9126 standardista, on FURPS:

- F - toiminnallisuus (functionality)
- U - käytettävyyys (usability)
- R - luotettavuus (reliability)
- P - suorituskky (performance)
- S - tuettavuus (supportability).

Laatumallien avulla voidaan priorisoida ominaisuuden tärkeys.

Toteutustapa on vapaa, määrittely ei puutu siihen. Määrittely tähtää siihen, että sen luettuaan kehittäjä ymmärtää, miten käyttötapauskuvaukset toimivat ja mi-

ten kaikki on muotoiltu. Tämä ei kuitenkaan toteudu juuri koskaan, mutta siihen pyritään.

Tekninen määrittely on arkkitehtuurisuunnittelun tuottama dokumentti. Sen tarkoituksena on kuvata, millainen rakenne järjestelmään tulee ja mitä siinä käytetään, esimerkiksi ohjelmointikielet, tietorakenteet ja rajapinnat. Tekninen määrittely usein jakaantuu moniin eri järjestelmän kerroksia kuvaaviin osioihin, jotta suunnitelman selkeys säilyy. (Haikala & Märijärvi 2004, s. 80–81.)

#### **3.2.4 Toteutus**

Toteutuksessa aletaan järjestelmää toteuttaa määrittelyjen pohjalta. Toteutuksen jälkeen ohjelmisto on osaltaan käyttövalmis.

Toteutukseen voidaan käyttää kahta eri mallia: top-down tai bottom-up. (Haikala & Märijärvi 2004, s. 38–48.)

Top-down -mallissa toteutus alkaa ylimmältä tasolta. Tämän jälkeen siirrytään toteuttamaan ylimmän tason alatasoja ja näin jatketaan, kunnes ohjelma on täysin toteutettu eli siirrytään suuremmasta aina pienempään ja tarkempaan. (Haikala & Märijärvi 2004, s. 38–48.)

Bottom-up -mallissa toteutetaan ensin pienet osat ja niitä yhdistelemällä koostaan suurempia osia, jolloin ohjelma alkaa hiljalleen muodostua. (Haikala & Märijärvi 2004, s. 38–48.)

#### **3.2.5 Testaus**

Testaus on erittäin tärkeä osa elinkaarta ja se viekin paljon aikaa. Sen tarkoituksena on löytää mahdollisimman monta virhettä, jotta ohjelman julkaisuversio on korkea laatu voidaan taata. Luvussa 5 käydään tarkemmin läpi ohjelmiston testaaminen. (Haikala & Märijärvi 2004, s. 38–48.)

#### **3.2.6 Julkaisu/käyttöönotto**

Ohjelmiston julkaistaan, kun kaikki siihen liittyvät tahot ovat tuotteeseen tyytyväisiä. Julkaisuun liittyy levityksen lisäksi myös ohjekirjojen teko sekä markkinointi.

Käyttöönotto tehdään tietylle asiakkaalle tehdylle ohjelmistolle eli on vastakohta julkisen ohjelmiston julkistukselle. Käyttöönotossa ohjelmisto asennetaan asiakkaan laitteille ja suoritetaan muut ohjelmiston toimimiseen vaikuttavat toimenpiteet. (Haikala & Märijärvi 2004, s. 38–48.)

### **3.2.7 Ylläpito**

Ylläpito on ohjelmiston käyttöönoton jälkeistä aikaa, jonka tarkoituksena on pitää ohjelmisto käyttökelpoisena ja korjata vielä mahdollisia virheitä sekä päivittää uusia ominaisuuksia, mikäli niitä on tarpeen kehittää. (Haikala & Märijärvi 2004, s. 38–48.)

### **3.2.8 Elinkaaren loppu**

Ohjelmiston elinkaaren loppu on vain sen tarpeettomaksi käyminen. Syitä siihen voi olla useita:

- ohjelmiston käyttötarve loppuu
- uusi käyttöympäristö, jossa ohjelmaa ei voida käyttää
- ohjelmisto sulautetaan johonkin toiseen ohjelmistoon
- keksitään parempi tapa ohjelman suorittamille toiminnoille.

(Haikala & Märijärvi 2004, s. 38–48.)

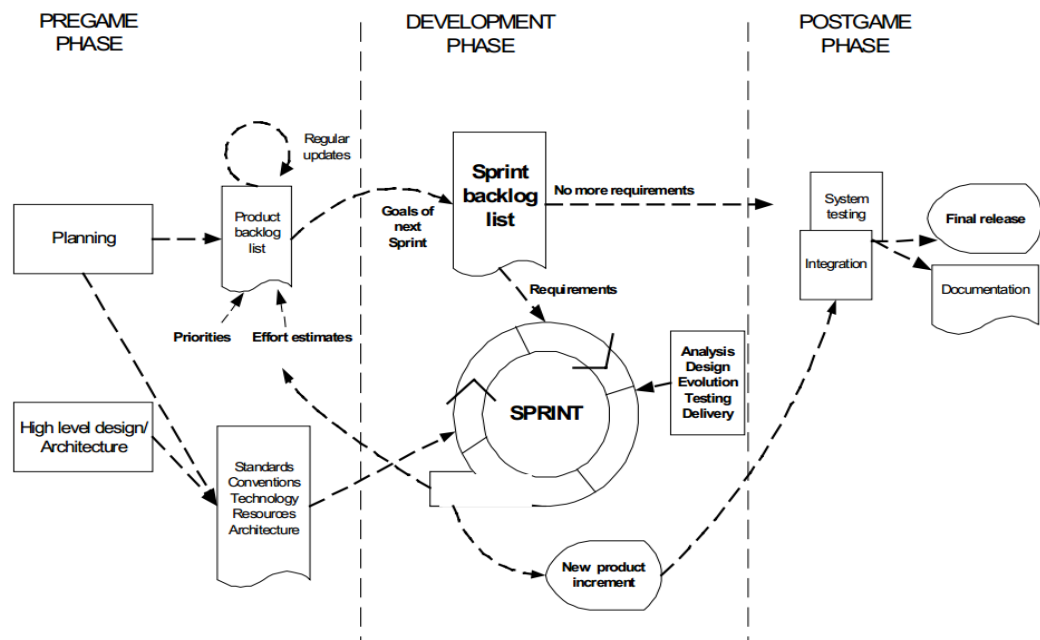
## **3.3 Ketterä ohjelmistokehitys**

Ketterät ohjelmistokehitysmenetelmät ovat syntyneet nykyisten ohjelmistokehitysvaatimusten vuoksi: tuote on saatava nopeasti markkinoille. Vesiputousmallin vastakohtina ketterät menetelmät ovat hyviä pienten ohjelmistoprojektien kehityksessä. Iteraatiot ovat lyhyitä, dokumentaatio on minimaalista ja suunnittelua ei tehdä pitkälle etukäteen, koska se saattaisi olla turhaa jatkuvien muutosten vuoksi. Ketterä-nimitys tulee kehitystyylin nopeasta muutokseen sopeutumisesta ja lyhyistä kehityssykleistä. Ohjelmasta saattaa tulla uusi versio joka kuukausi. (Haikala & Mikkonen 2011, s. 43–46.)

### **3.3.1 Scrum**

Scrum on suosittu, pieniin projekteihin sopivampi ohjelmistokehitysmenetelmä. Scrum-menetelmän ideana on monien alueiden eri ammattilaisista koostuvan

tiimin eteneminen yhtenä joukkona tavoitetta kohti. Kuva 6 kuvaa Scrumia. (Scrum.)



Kuva 6. Scrum (Abrahamsson, Salo, Ronkainen & Warsta 2002, s.28)

Scrum toimii nopeissa iteraatioissa, sprinteissä, kuvan Kuva 6 mukaisesti, jotka kestävät enintään kuukauden. Ensin luodaan tuotteen vaatimuslista (Product Backlog), joka on järjestetty lista kaikesta, mitä tuotteessa saatetaan tarvita. Jokaisen sprintin jälkeen pidetään vanhan sprintin tarkastelupalaveri, jossa tarkastellaan, mitä saatiin aikaiseksi ja miten työt sujuivat, sekä uuden sprintin suunnittelupalaveri, jossa jaetaan työlistat seuraavaan iteraatioon (Sprint Backlog). Ohjelma kehittyy jokaisessa sprintissä hieman lisää ja paremmaksi. Scrumin toimintatapoihin kuuluvat myös päiväpalaverit, jotka pidetään Scrum-mestarin sekä kehitystiimin kesken. Näissä jokainen tiimin jäsen vastaa kolmeen kysymykseen:

- Mitä olet tehnyt edellisen palaverin jälkeen?
- Mitä aiot tehdä seuraavaksi?
- Onko tiedossa esteitä, jotka hidastavat työtäsi?

(Scrum.)

Käytännössä päiväpalavereja ei kuitenkaan pidetä välttämättä joka päivä, vaan esimerkiksi viikon välein. (Scrum.)

Scrum-kehityksessä rooleja on kolme: tuoteomistaja, kehitystiimi sekä Scrum-mestari. (Haikala & Mikkonen 2011, s. 46–51.)

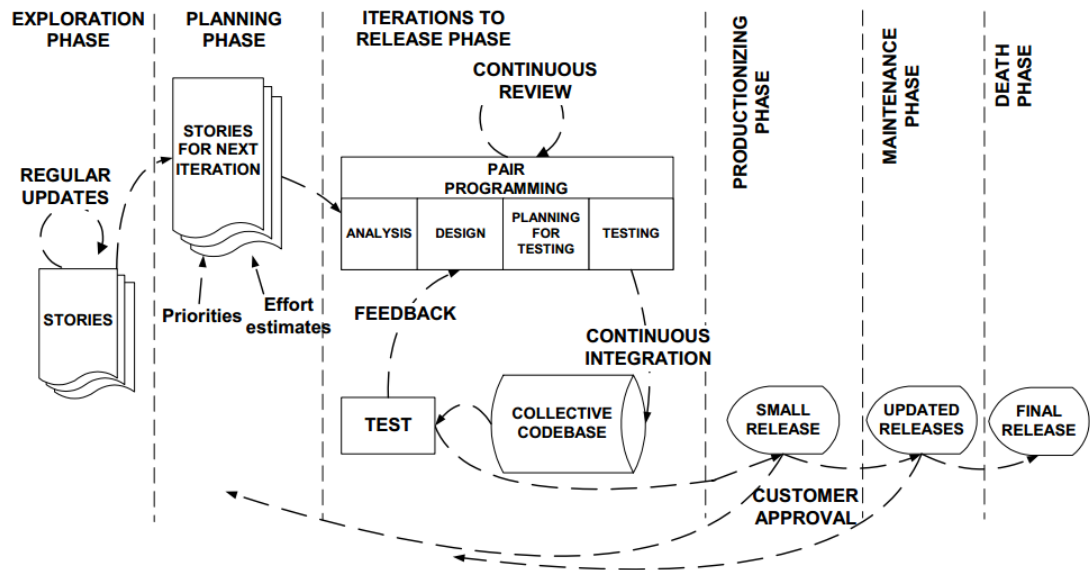
Tuoteomistaja on vastuussa kehityksen tärkeysjärjestyksestä eli mikä ominaisuus on seuraavaksi tärkein ja tuotteen vaatimusten määrittelystä. Tuoteomistaja edustaa myös asiakkaita ja voi toimia heidän yhteyshenkilönään. Tuoteomistajan tärkein tehtävä sprinttipalavereissa on määrittää seuraava kehitysjonon vaihe. (Haikala & Mikkonen 2011, s. 46–51.)

Kehitystiimin koko on pieni, joten tiimin jäsenten ammattitaidon on oltava laaja-alaista, jotta jokaisella ohjelmistokehitystyön osa-alueella on tietotaitoa. Tiimejä voi olla useita, jolloin Scrum soveltuu myös suurien ohjelmistoprojektien kehitykseen. Kehitystiimi on itse vastuussa työnjaosta sekä töiden etenemisestä. (Haikala & Mikkonen 2011, s. 46–51.)

Scrum-mestarin tehtävänä on vain huolehtia, että kehitystiimi noudattaa Scrumin periaatteita. Mestari ei kerro, mitä tehdä, vaan ainoastaan varmistaa, että asiat tehdään oikealla tavalla. (Haikala & Mikkonen 2011, s. 46–51.)

### **3.3.2 Extreme programming, XP**

XP on ketterä ohjelmistokehitysmenetelmä, jonka tarkoituksena on parantaa ohjelmiston kehitystyön laatua sekä reaktiokykyä muuttaa ohjelmaa asiakkaan vaatimusten vaihtuessa. Kuva 7 esittää XP-menetelmän elinkaarta. (Don Wells. XP.)



Kuva 7. Extreme programming (Abrahamsson ym. 2002, s.19)

Ketterän menetelmän ideana on julkaista väliversioita lyhyin väliajoin (Kuva 7), mikä vuorostaan parantaa tuottavuutta ja tarjoaa tarkistuspisteitä, joihin voi helposti liittää uusia asiakkaan vaatimuksia. (Don Wells. XP.)

Pariohjelmoinnissa ohjelmointi tehdään pareittain eli toinen ohjelmoi ja toinen tarkistaa ohjelman koodia antaen palautetta välittömästi. Tämä menetelmä vähentää virheiden muodostumista huomattavasti. (Don Wells. XP.)

Testivetoinen kehitys (TDD) on tekniikkaa, jossa luodaan ensin testitapaus, jonka jälkeen muokataan ohjelmaa niin, että ohjelma läpäisee testin. (Don Wells. XP.)

Edellä mainitut ominaisuudet sekä ominaisuuden ohjelmointi, yksinkertainen ja selkeä koodi, muutosten odottaminen ja niihin valmistautuminen sekä usein toistuva kommunikaatio muiden ohjelmoijien ja asiakkaan kanssa ovat tärkeä osa XP-menetelmää. (Don Wells. XP.)

XP pyrkii vähentämään vaatimusten muutosten kuluja lyhyillä kehityskierroksillaan. Muutokset ovat haluttu osa ohjelmiston kehitystä ja niihin varaudutaan. (Don Wells. XP.)

## **XP-toiminnot**

XP:lla on neljä perustoimintoa ohjelmistokehitysprosessissaan, jotka ovat ohjelmointi, testaus, kuuntelu ja suunnittelu. (Don Wells. XP.)

Ohjelmointi on tärkeä osa ohjelmistokehitystä. Ilman koodia ei ole toimivaa tuotetta. Ohjelmoinnilla voi kuitenkin myös yrittää selvittää parhaimman ratkaisun ongelmiin tai välittää ajatuksiaan paremmin toisille ohjelmoijille. Esimerkiksi jonkin vaikean ongelman voi ehkä pystyä selittämään yksinkertaisella koodilla muille, jotka voivat siten auttaa ratkaisemaan ongelman. (Don Wells. XP.)

XP:n ajatuksena testauksessa on, että jos pienellä testauksella löytää muutamia virheitä, suurella määrällä testejä löytää suuren määrän virheitä. (Don Wells. XP.)

XP:ssä on kolme eri testaustapaa, jotka ovat yksikkötestaus, hyväksyttämistestaus ja integraatiotestaus. Nämä tavat esitellään tarkemmin luvussa 4. (Don Wells. XP.)

Ohjelmoijien on kuunneltava asiakkaan vaatimuksia ja ymmärrettävä ne tarpeeksi hyvin, jotta he voivat keskustella asiakkaan kanssa, miten kyseessä olevan asian voi toteuttaa ja voiko sitä edes toteuttaa. (Don Wells. XP.)

Vaikka XP onkin enemmän vain eteenpäin liikkumista, täytyy suunnitteluun silti panostaa, koska lopulta tulee eteen kohta, joka tarvitsee suunnittelua. Hyvän suunnittelun ansiosta näiltä tilanteilta vältytään ja sitä myöten ohjelman toimivuus säilyy pidempään muutoksista huolimatta. (Don Wells. XP.)

## **Arvot**

XP sisältää viisi eri arvoa toiminnassaan, jotka ovat kommunikaatio, yksinkertaisuus, palaute, rohkeus ja kunnioitus. (Don Wells. XP.)

Perinteisissä menetelmissä tiedon välittäminen asiakkaalta kehittäjälle tapahtuu dokumentaatiolla. XP:ssa asiakas on todella usein läsnä, joten kehitysryhmä saa jatkuvaa palautetta ja näin varmistutaan myös paremmin, että kehittäjät jakavat näkemyksensä asiakkaan kanssa. (Don Wells. XP.)

XP:ssa ohjelmointi aloitetaan yksinkertaisimmasta ratkaisusta. Uudet toiminnallisuudet lisätään aina myöhemmin eli keskitytään ohjelmoimaan sitä, mitä tarvitaan tänään eikä vasta tulevaisuudessa. Näin menetellen kuitenkin voi käydä niin, että uusi ominaisuus vaatii vanhan koodin muuttamista, joskus paljonkin, jolloin työtaakka lisääntyy. Tälle on kuitenkin vastaväite, että näin ei kulu aikaa sellaisten ominaisuuksien ohjelmointiin ja suunnitteluun, joita ei tulevaisuudessa ehkä tarvitakaan. (Don Wells. XP.)

Yksinkertaisen koodin välittämä tieto myös välittyy helpommin ymmärrettävänä muille ohjelmoijille. (Don Wells. XP.)

XP:ssa palaute on erittäin tärkeää kehityksen kannalta. Palautetta saadaan järjestelmästä yksikkötestien avulla ja asiakkaalta hyväksyttämistestien avulla. Asiakas saa palautetta kehitysryhmän toiminnasta suunnitteluvaiheessa. (Don Wells. XP.)

Rohkeus liittyy XP:ssa siihen, että koodataan tälle päivälle, jolloin huomisen vaatimukset eivät pelota, vaan ohjelmoija pystyy tekemään koodin juuri haluamallaan tavalla. Rohkeus on myös sitä, kun uskaltaa heittää koodia pois ja kun ei anna periksi ongelman ollessa ylitsepääsemätön. (Don Wells. XP.)

Kunnioitus liittyy ryhmän työskentelyyn. Yksikään ohjelmoija ei saa omilla muutoksillaan tuhota toisen ohjelmoijan tekemää työtä. Edellä mainitut neljä arvoa johtavat kunnioitukseen ryhmän kesken. (Don Wells. XP.)

## **Periaatteet**

XP:ssa on muutamia konkreettisempia periaatteita kuin edellä mainitut arvot, jolloin niistä saa helpommin kiinni. Ne ovat palaute, yksinkertaisuus ja muutosten omaksuminen. (Don Wells. XP.)

Palaute on hyödyllisintä, kun sitä tulee usein ja nopeasti. Palautteen nopeus on erittäin tärkeää oppimisen kannalta, mitä nopeammin palautteen saa, sitä paremmin kehittäjä oppii ja voi muuttaa asioita. (Don Wells. XP.)

Kommunikaation kautta tuleva palaute asiakkaalta tulee useammin XP-menetelmässä. Palautetta antamalla asiakas voi ohjata kehitysprojektin kulkua



tarvittaessa. Näin ollen myös kehittäjän tekemä suunnitteluvirhe huomataan nopeammin, ennen kuin hän ehtii käyttämään liikaa aikaa sen toteuttamiseen. (Don Wells. XP.)

Yksikkötestit antavat palautetta järjestelmän toiminnasta, ja on tärkeää suorittaa kaikkien osien, myös muidenkin kuin juuri kirjoitetun koodin, testit. Näin ollen huomataan myös virheet, jotka ilmenevät ohjelman muissa osissa, jolloin kehittäjä pystyy reagoimaan niihin ja tekemään tarvittavia muutoksia. Ajettaessa kaikki testit samaan aikaan varmistutaan, etteivät tällaiset virheet jää jopa julkaisuversioon asti, jolloin niiden alkuperän etsiminen on vaikeaa sekä ratkaisu työlästä. (Don Wells. XP.)

Oletetaan, että jokaisen ongelman ratkaisu on todella yksinkertainen. XP ei tue jatkoa varten ohjelmointia, vaan ohjelmoidaan sitä hetkeä varten. Pienillä muutoksilla ja etenemisillä saadaan suuria tuotteita ja näin myös asiakas on enemmän mukana kehityksessä. (Don Wells. XP.)

Muutosten tullessa eteen niitä vastaan ei taistella vaan ne omaksutaan ja niiden mukaan jatketaan. (Don Wells. XP.)

## **Kritiikki**

XP on saanut paljon arvostelua osakseen, muun muassa seuraavaa:

- voidaan käyttää lypsämään rahaa asiakkaalta määrittelemättömän lopputuotteen takia
- rakenteen ja dokumentoinnin puute
- toimii vain kokeneilla kehittäjillä
- jatkuvat tapaamiset maksavat asiakkaalle paljon
- melkein mahdotonta arvioida, kauanko tietyssä osassa menee, koska kukaan ei tiedä, millainen valmiista tuotteesta tulee
- ominaisuusmäärä voi kasvaa liikaa, koska määrittely ei ole tarkka, jolloin projekti leviää liian suureksi työryhmälle.

(Don Wells. XP.)

## 4 Ohjelmiston testaaminen

Ohjelmiston testaus on erittäin tärkeää, koska se antaa tietoa testattavan tuotteen tai palvelun laadusta. Testauksen hyötyjä ovat myös taloudelliset säästöt ja ajalliset hyödyt. (Haikala & Märijärvi 2004, s. 283–299.)

Ulkopuolisella testaajalla on myös objektiivinen näkökulma tuotteeseen tai palveluun, jolloin testaustulokset ovat riippumattomia toteuttajien intresseistä. Näin ollen ohjelman tekijä saa suoraa tietoa siitä, täyttääkö ohjelma sille asetetut vaatimukset toiminnallisuudeltaan ja suorituskyvyltään ja onko sen käytettävyys hyvällä vai huonolla tasolla sekä tietenkin onko siinä virheitä, jotka tarvitsevat korjausta. (Haikala & Märijärvi 2004, s. 283–299.)

Ohjelmistotestauksen tavoitteena on löytää virheitä ja vikoja, jotka aiheuttavat häiriöitä ohjelman suorituksen aikana. Testaus ei kuitenkaan takaa, että kaikki virheet löytyisivät vaan kertoo, että ohjelma toimii ainakin testatussa ympäristössä mallikkaasti. Tavoitteeseen pääsemiseksi testaus tarvitsee ohjelman koodin tutkimista sekä sen suorittamista erilaisissa ympäristöissä ja tilanteissa riittävän monta kertaa. Ulkopuolinen testaaja on parempi kuin ohjelmistoprojektin sisäinen objektiivisuutensa vuoksi. (Haikala & Märijärvi 2004, s. 283–299.)

Ohjelmistoviat ilmenevät yleensä suoraan koodissa olevan virheen vuoksi, josta syntyy häiriö ohjelmaa suoritettaessa. Testauksen vajavaisuuden vuoksi suoritamattomassa koodissa oleva virhe ei ilmene ennen kuin ympäristö tai testaus-tapa vaihtuu. Tämä taas johtaa saattaa johtaa häiriöön, kun suorituspaiikka tai suoritusalusta vaihtuu. Tällainen virhe voi olla kohtalokas ohjelman suoritukselle ja voi johtaa useiden muiden häiriöiden syntymiseen. Häiriöiden ilmetessä ne täytyy korjata niiden vaatimalla tavalla. (Haikala & Märijärvi 2004, s. 283–299.)

Kaikki häiriöt eivät kuitenkaan ole yksinkertaisia ja johdu koodissa olevista virheistä. Syitä voi olla useita:

Tuntemattomat vaatimukset, joita ei ole osattu ottaa huomioon ja jotka aiheuttavat suuren virheen suorituksen aikana. (Haikala & Märijärvi 2004, s. 283–299.)

Yhteensopivuus muiden ohjelmien kanssa, esimerkiksi uuden käyttöjärjestelmän tai selaimen kanssa, mikä johtaa ohjelman toimimattomuuteen. Helpoin

esimerkki on Windows- ja Linux-alustoille erikseen tehtävät ohjelmat. Uusimmissa versioissa saattaa olla lisätty/poistettu joitain ominaisuuksia, joita testattava ohjelma tarvitsee, jolloin se ei toimi vanhemmissa versioissa. (Haikala & Märijärvi 2004, s. 283–299.)

#### **4.1 Testausmetodit**

Staatinen testaus on ohjelman testausta ilman sen suorittamista esimerkiksi katselmoimalla, tarkastelemalla tai toisen ohjelman kääntäjän avulla. (Kautto 1996.)

Dynaaminen testaus on ohjelman koodia suorittamalla tapahtuvaa testausta ja se jaetaan kolmeen eri luokkaan: mustalaatikkotestaus, harmaalaatikkotestaus sekä lasilaatikkotestaus. (Kautto 1996.)

Mustalaatikkotestauksessa ohjelma kuvitellaan mustana laatikkona, josta ei tiedetä, mitä se sisältää. Virheitä yritetään etsiä yleensä toiminnallisen määrittelyn pohjalta tehtyjen testitapauksien avulla ja loppukäyttäjän näkökulmasta. Hyvä puoli mustalaatikkotestauksessa on sen objektiivisuus, mutta huonona sen tietämättömyys ohjelmasta, koska ei tiedetä, miten mikäkin asia toimii, joten testaaja suorittaa myös monta turhaa testitapausta. Mustalaatikkotestausta voi suorittaa kuka tahansa. (Kautto 1996.)

Harmaalaatikkotestauksessa testaaja pääsee tutustumaan ohjelman rakenteisiin ja koodiin silloin, kun hän suunnittelee testitapauksia. Itse testaaminen tapahtuu kuitenkin mustalaatikkotasolla. (Kautto 1996.)

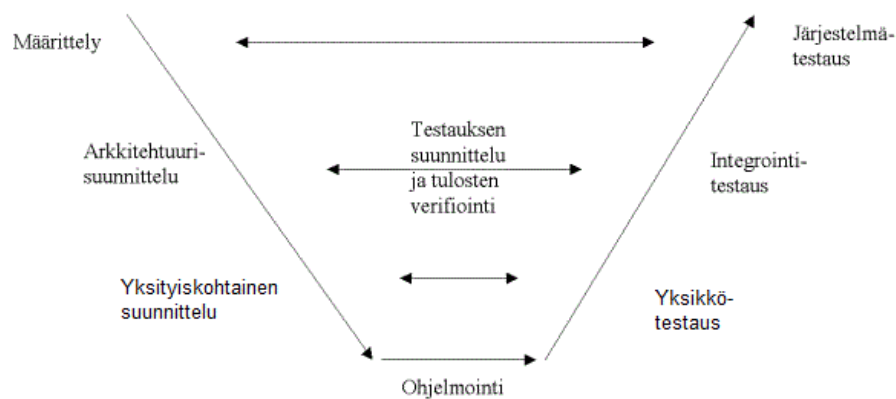
Lasilaatikkotestauksessa testaaja tietää ohjelman rakenteen sekä koodin. Lasilaatikkotestauksen tarkoituksena on testata nimenomaan rakenteiden toimivuutta eli on vastakohta mustalaatikkotestaukselle. Testitapauksien tekoon vaaditaan tässä testausstyypissä hyvä tietämys ohjelmasta sekä ohjelmointitaitoja. (Kautto 1996.)

Visuaalisessa testauksessa ohjelman tekijä sekä testaajat ovat samassa tilassa ja koko testaus tilanne videoidaan myöhempää analysointia varten. Tämä mahdollistaa virhetilanteiden sekä niihin johtaneiden toimenpiteiden tallentamisen. Visuaalinen testaus on hyvä myös siksi, että kehittäjä huomaa virheen heti ja

pystyy analysoimaan mahdollisen alkuperän ilman, että virhettä tarvitsee toistaa tai selittää kirjoittamalla. Tämä testaustapa on kätevä ketterissä ohjelmistotuotantomenetelmissä, joissa kommunikaatio on suuressa osassa projektin etene-  
misen kannalta. (Kautto 1996.)

## 4.2 Testaustasot

V-mallia käytetään usein havainnollistamaan ohjelman kehitystyön ja testauk-  
sen suhdetta (Kuva 8). (Kautto 1996.)



Kuva 8. V-malli (Haikala & Mikkonen 2011, s. 207)

Yksikkötestaus tarkoittaa yksittäisen luokan tai moduulin testaamista. Tämä ta-  
pahtuu white box -testauksella ohjelman kehittämisen aikana. Luokan toimintaa  
verrataan yksityiskohtaisen suunnittelun ja arkkitehtuurisuunnittelun tuloksiin  
(Kuva 8). (Kautto 1996.)

Integraatiotestauksessa suoritetaan komponenttien ja ohjelmistosuunnittelun  
välisten rajapintojen testausta. Integraatiotestaus suoritetaan aina, kun uusi  
komponentti ollaan liittämässä ohjelmistoon. (Kautto 1996.)

Järjestelmätestauksessa tarkastellaan koko järjestelmää ja verrataan sen tulok-  
sia ohjelmiston vaatimusmäärittelyyn, toiminnalliseen määrittelyyn, käyttöohjee-  
seen ja muihin asiakasdokumentteihin. Järjestelmätestauksen suorittajien täytyy  
olla mahdollisimman riippumattomia ohjelmiston kehityksestä. (Kautto 1996.)

Hyväksyttämistestauksessa valmis ohjelma toimitetaan asiakkaalle hyväksytet-  
täväksi eli asiakas testaa onko tyytyväinen ohjelmaan. (Kautto 1996.)

### 4.3 Testaustyytit

Testaustyyppit on myös monenlaisia (Kautto 1996).

**Asennustestaus** on testi, jossa varmistetaan, ett  vaadittava j rjestelm  on asennettu oikein ja toimivasti asiakkaan laitteistolle.

**Yhteensopivuustestauksella** varmistutaan, ett  ohjelma toimii valitulla alustalla ilman muiden ohjelmien tai alustan itse aiheuttamia h iri it .

**J rkevyystestauksessa** m  ritet    onko j rkev   jatkaa testausta.

**Aloitustestauksessa** ohjelmaa yritet    k ytt   mahdollisimman pienill   k  mennoilla, jolloin huomataan, onko ohjelmassa mit    perustavanlaatuista vikaa, joka est isi sen k yt  .

**Regressiotestauksella** tarkoitetaan testausta, joka suoritetaan muutosten j lkeen. Sen tarkoituksena on l ytt   ohjelmaan palanneita ohjelmointivirhe tai ongelmia. Regressiotestausta kannattaa suorittaa s   nn  llisesti, jotta virheet huomattaisiin heti, eik  vasta julkaisun j lkeen. N in ollen korjauskulut j  v   huomattavasti pienemmiksi.

**Alpha-testaus** on simuloitua tai varsinaista testausta mahdollisten k ytt jien toimesta ohjelman kehitt j   luona. Alpha-testauksen tarkoituksena on toimia yrityksen sis isen  hyv ksytt mistestauksena ennen kuin siiryt    ohjelmiston kehityksess  seuraavaan vaiheeseen.

**Beta-testaus** on seuraava vaihe Alpha-testauksen j lkeen. Beta-testauksessa ohjelman silloista versiota jaetaan valituille henkil  ille/yrityksille, jotka voivat testata ohjelmaa omassa ymp rist ss    ja antaa testauksesta palautetta. Mit  monimuotoisempi testausryhm  on, sit  varmemmin l ydet    viat, koska jokainen henkil  k ytt   ohjelmaa omalla tavallaan. Beta-testaus voi olla my   avoin, jolloin testausryhm  kasvaa ja sit  my ten my   palaute ja vikojen l ytymisen mahdollisuus kasvaa.

**Toiminnallisessa testauksessa** testataan ohjelman jotakin osaa vastaten usein kysymyksiin ”voiko n in tehd  ” tai ”toimiiko t    ominaisuus”. Toiminnallinen testaus perustuu vaatimusm  rittelyyn. Positiivisilla testitapauksilla var-

mistetaan, että ohjelma tekee sen mitä on vaadittu. Negatiivisilla testitapauksilla varmistetaan, että ohjelma toimii järkevästi myös poikkeustilanteissa.

**Destruktiivisen testauksen** tarkoituksena on tuhota ohjelma, eli saada ohjelma itse tai sen alijärjestelmä kaatumaan. Testaus varmistaa, että ohjelma toimii ongelmitta vaikka sille syötetään vääriä tai odottamattomia arvoja. Näin ollen varmistetaan ohjelman lujuudesta ja sen virheensietokyvystä.

**Suorituskykytestaus** tehdään järjestelmän tai alijärjestelmän reagointikyvyn ja vakauden testaamiseksi tietyn työtaakan alla. Sillä voidaan myös tutkia muita ohjelman ominaisuuksia, kuten skaalaavuutta, luotettavuutta ja resurssien käytön määrää.

**Käytettävyydestausta** tarvitaan käyttöliittymän toimivuuden ja helppokäyttöisyyden testaamiseen.

**Saavutettavuustestauksessa** testataan ohjelman saavutettavuutta eli, kuinka helposti sitä voi lähestyä sekä sen käytön esteettömyyttä. Testauksen tarkoituksena on määrittää kuinka helppoa erilaisten ihmisten on käyttää ja saada käyttöönsä testattava ohjelma.

**Tietoturvatestauksen** tarkoituksena on ohjelman luottamuksellisen tiedon saavuttamattomuuden tason määrittäminen eli selvitetään, kuinka hyvin tiedot on suojattu ulkopuolisilta tunkeutujilta, kuten hakkereilta ja haittaohjelmilta.

**Kansainvälistämisen ja lokalisoinnin testaamisen** ideana on saada selville, kuinka ohjelma suoriutuu toiminnoistaan kielen tai aikavyöhykkeen vaihtuessa. Eri kielten käännökset täytyy myös testata mahdollisten lokalisointivirheiden varalta.

**Kehitystestaus** on ohjelman kehityksen prosessi, eli osa itse kehitystä, jonka tarkoituksena on eliminoida rakenteellisia virheitä ennen kuin koodi menee tarkistettavaksi kehityksen seuraavaan vaiheeseen. Kehitystestaus on hyvä tapa saada lopullisesta ohjelmasta laadukkaampi.

#### **4.4 Testaustyökalut**

Testaustyökaluja on monenlaisia erilaisiin tarkoituksiin. Niiden päämääräinen tarkoitus suorittaa testaajien kehittämiä testejä nopeasti ja aina samalla tavalla. Näin ollen testaajien ei itse tarvitse kuluttaa aikaa pitkiin testauksiin, vaan ne voidaan vain suorittaa työkalulla. (Kautto 1996.)

Ketterissä menetelmissä ja iteroivissa ohjelmistokehitystyöissä testausvälineet ovat välttämättömiä päivittäin regressiotestauksen takia. On myös tilanteita, joita ei voi manuaalisesti suorittaa, esimerkiksi samanaikaisuustestaus ja kuormitustestaus. (Kautto 1996.)

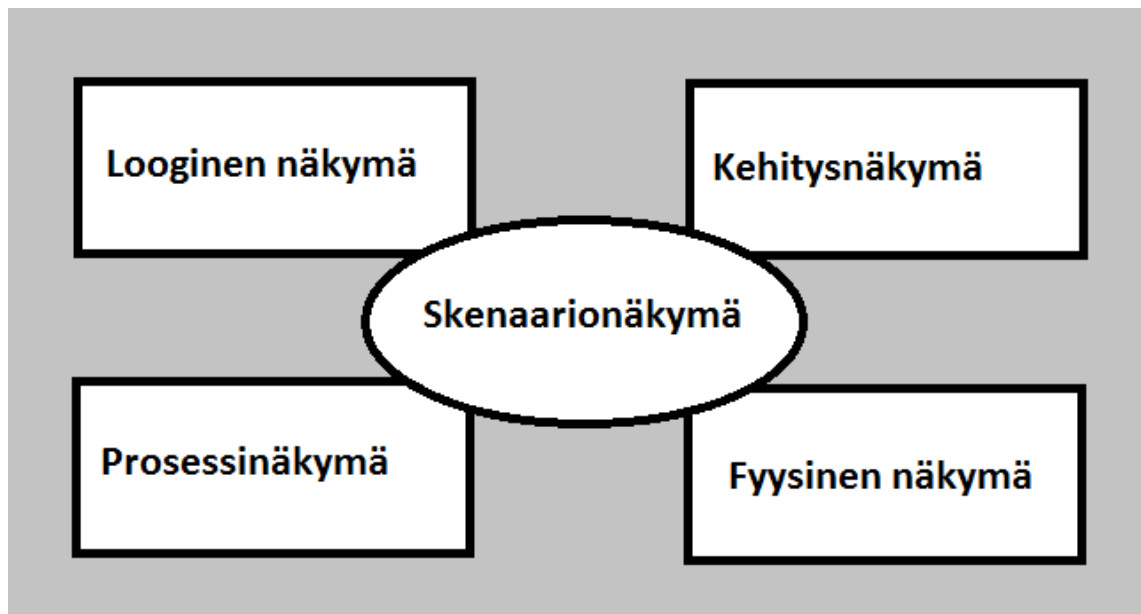
Testaustyökalujen käyttö on erittäin kannattavaa pidemmällä aikavälillä, jolloin ohjelmistopäivityksiä tulee useampia ja niiden testausta voi suorittaa työkaluilla. Tämä takaa toimivan ohjelman myös päivitysten jälkeen. Ainoa ongelma on päättää, mitä automatisoidaan ja milloin. (Kautto 1996.)

### **5 Arkkitehtuuri**

Arkkitehtuuri, tässä tapauksessa tietojärjestelmän arkkitehtuuri, kuvaa järjestelmän rakenneosat, niiden näkyvät ominaisuudet sekä niiden väliset yhteydet ja riippuvuudet. Arkkitehtuuri on runko järjestelmän suunnittelulle sekä toteutukselle ja ohjaa rakenteiden kehitystä järjestelmän elinkaaren ajan.

#### **5.1 Ohjelmistoarkkitehtuuri**

Ohjelmistoarkkitehtuuri kuvaa ohjelmiston alijärjestelmiä ja komponentteja sekä niiden välisiä yhteyksiä. Komponentit voivat olla pieniä koodin osia, kuten moduuleja, tai suuria osajärjestelmiä, kuten tietokannan hallintajärjestelmä. Kuva 9 esittää usein ohjelmistoarkkitehtuurin kuvaamiseen käytettävää 4+1-mallia. (Koskimies & Mikkonen 2005, s. 35–37.)



Kuva 9. 4+1 Kruchten (Wikipedia. Kruchten 4+1-kuva)

Ohjelmistoarkkitehtuuri kuvataan eri näkökulmista niin sanotulla 4+1-mallilla (Kuva 9). Siinä ohjelmistoarkkitehtuuri jaetaan viiteen rinnakkaiseen näkökulmaan, jotka ovat looginen näkymä, kehitysnäkymä, skenaarionäkymä, prosessinäkymä ja fyysinen näkymä. (Koskimies & Mikkonen 2005, s. 35–37.)

Looginen näkymä (logical view) kuvaa järjestelmän loogisia rakenneosia kuten luokkia ja olioita sekä niiden välisiä yhteyksiä ja velvollisuuksia. Looginen näkymä soveltuu hyvin yksityiskohtaisen suunnittelun pohjaksi, ja se on olennainen osa kaikille järjestelmille. Sen kuvausvälineinä käytetään usein luokkakaavioita, oliokaavioita ja yhteistyökaavioita. (Koskimies & Mikkonen 2005, s. 35–37.)

Kehitysnäkymä (development view) kuvaa ohjelman ohjelmistoteknistä rakennetta. Siinä pilkotaan ohjelmisto erillisiin osiin, jotka voidaan toteuttaa erillisinä yksikköinä. Eri osien yhteenliittyminen voidaan kuvata esimerkiksi pakkaus-, olio- tai luokkakaavioilla. Kehitysnäkymän käyttäminen on erittäin tärkeää etenkin suuria ohjelmistoja tehdessä. (Koskimies & Mikkonen 2005, s. 35–37.)

Skenaarionäkymä (scenario view) kuvaa järjestelmän rajapinnan ympäristöönsä tarkastelemalla, kuinka järjestelmä on vuorovaikutuksessa sen ulkopuolisiin käyttäjiin ja muihin järjestelmiin. Koska skenaarionäkymä kuvaa järjestelmän



keskeistä toiminnallisuutta, on se usein lähtökohtana muiden näkymien muodostamiselle. (Koskimies & Mikkonen 2005, s. 35–37.)

Prosessinäkymä (process view) kuvaa, miten järjestelmän toiminta on jaettu loogisiin prosesseihin ja miten ne kommunikoivat keskenään. Prosessinäkymää tarvitaan varsinkin järjestelmiin, joissa vaaditaan rinnakkaisuutta ja sillä voidaan hyvin arvioida järjestelmän suorituskykyä ja skaalaavuutta. (Koskimies & Mikkonen 2005, s. 35–37.)

Fyysinen näkymä (physical view) kuvaa järjestelmässä tarvittavat fyysiset artefaktit, miten artefaktit ovat yhteydessä toisiinsa, millaisista osista järjestelmä koostuu ja mihin prosessointiyksiköihin mitään sijoitetaan. (Koskimies & Mikkonen 2005, s. 35–37.)

Keskeinen osa ohjelmistoarkkitehtuuria määriteltäessä on ohjelmiston jakaminen osiin. Se auttaa hallitsemaan kokonaisuutta, mahdollistaa valmiiden osien käyttämisen sekä helpottaa ratkaisun ymmärtämisessä. (Koskimies & Mikkonen 2005, s. 35–37.)

## **5.2 Suunnittelumallit**

Suunnittelumallit ovat yleisiä ratkaisuja ohjelmistojen suunnittelun yleisimpiin ongelmiin. Niissä kiteytyy vanhojen suunnittelumestarien kokemus ja tietotaito. Ensimmäiset suunnittelumallit alkoivat esiintyä jo 1980-luvun lopulla, mutta todellisen läpimurron teki ”Gang of Four”-kirjan (Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns) julkaisu vuonna 1995. Siihen oli kerätty monia vielä nykyäänkin käytettäviä suunnittelumalleja. (Koskimies & Mikkonen 2005, s. 102–103.)

Suunnittelumalleja on useita ja ne ovat nimettyjä, jolloin niihin voidaan viitata helposti ja kaikki tunnistavat, mistä puhutaan. Suunnittelumallit koostuvat kolmesta osasta: ongelma, ongelmayhteys ja ratkaisu. (Koskimies & Mikkonen 2005, s. 102–103.)

Ongelman tulee olla yleinen suunnitteluongelma, eikä se saa olla sidoksissa tiettyyn ominaisuuteen, kuten ohjelmointikieleen, ja sen tulee ilmentyä useissa eri järjestelmissä. (Koskimies & Mikkonen 2005, s. 102–103.)

Ongelmayhteys kertoo millaisissa tilanteissa suunnittelumallia voi soveltaa ja myös asettaa ratkaisulle tiettyjä vaatimuksia liittyen suunnittelun ominaisuuksiin. (Koskimies & Mikkonen 2005, s. 102–103.)

Ratkaisun, kuten koko suunnittelumallin, on oltava yleinen ja se on oltava kuvailtavissa yleisesti tunnetuilla tavoilla, esimerkiksi UML. Ratkaisut täyttävät niiden vaatimukset, mutta samalla usein heikentävät toista ominaisuutta, esim. suorituskkyä. (Koskimies & Mikkonen 2005, s. 102–103.)

### **5.2.1 Antisuunnittelumallit**

Antisuunnittelumallit, tai vastasuunnittelumallit, ovat kuten suunnittelumallitkin. Ne esittävät ratkaisun yleiseen ongelmaan. Erona on se, että ne esittävät huonon ratkaisun; tavan, jolla ei kannata tehdä. Antisuunnittelumallit myös havainnollistavat ratkaisun seuraukset, jotka ilmenevät epätoivottuina ominaisuuksina myöhemmin ohjelmiston elinkaareissa. (Koskimies & Mikkonen 2005, s. 107–108.)

Antisuunnittelumalleissa voi olla liitettynä parempi ratkaisu käsillä olevaan ongelmaan, eli siis oikea suunnittelumalli. Näin ollen se voi olla yhtä hyödyllinen kuin varsinainen suunnittelumalli; se auttaa löytämään ohjelmistosta kohtia, jotka heikentävät ohjelmiston laatua. (Koskimies & Mikkonen 2005, s. 107–108.)

Antisuunnittelumalleja on myös useita, joskin niiden sisältö on hieman enemmän soveltamista vaativaa, sillä oikeita ratkaisuja on usein yksi tai kaksi, kun taas huonoja ratkaisuja voi olla moniakin tiettyyn ongelmaan. (Koskimies & Mikkonen 2005, s. 107–108.)

### **5.2.2 Hyödyt ja haitat**

Suunnittelumallien antamia hyötyjä on useita:

- kokeneempien suunnittelijoiden tietotaito välittyy nuorille suunnittelijoille
- kommunikoinnin edistäminen, sillä ongelmia tai niiden ratkaisuja voi selittää yhdellä sanalla: suunnittelumallin nimellä.

(Koskimies & Mikkonen 2005, s. 116–119.)

Suunnittelumalleilla on kuitenkin huonoja puolia:

- parantavat jotain laatuominaisuutta, mutta usein heikentävät jotain toista
- monimutkaistavat arkkitehtuuria lisäämällä komponenttien, luokkien ja rajapintojen määrää
- käytön jälkien häviäminen, koska ohjelmointikielet eivät niitä tue, jolloin dokumentoinnin tärkeys kasvaa uudelle tasolle.

(Koskimies & Mikkonen 2005, s. 116–119.)

### **5.3 Laitteistoarkkitehtuuri**

Laitteistoarkkitehtuuri kuvaa järjestelmän kokonaisuuden ja sen eri osien sijoittumisen järjestelmässä, eli käydään läpi mitä laitteita, käyttöjärjestelmiä ja verkko- ja koratkaisuja tullaan käyttämään. Siinä käydään myös läpi eri osien toteutus- ja tiedonvälityspäätteet. Laitteistoarkkitehtuurilla pyritään varmistamaan laitteiden yhteensopivuus ja skaalaavuus. (Nyström 2005.)

### **5.4 Ajonaikainen arkkitehtuuri**

Ajonaikaista arkkitehtuuria tarvitaan tukemaan sovellusta ja sovellusarkkitehtuuria. Ajonaikaisessa arkkitehtuurissa otetaan huomioon ohjelmiston teknologia, johon kuuluu verkko, laitteisto ja niin edelleen, päätyen teknisiin ohjelmistopalveluihin, jotka voivat olla sovellusriippumattomia ja, joita ilman sovellus ja sovellusarkkitehtuuri eivät toimi. (Nyström 2005.)

Mitä paremmalle tasolle ajonaikainen arkkitehtuuri saadaan, sitä yksinkertaisempaa sovelluskehitys on ja käytännössä se ilmenee niin, että koodia tarvitsee kirjoittaa vähemmän. Nykyään kaksi suosituinta perustaa, joille ajonaikainen arkkitehtuuri rakennetaan ovat Java ja .NET, mutta nekin eivät vielä kata kaikkia tarpeita. (Nyström 2005.)

#### **5.4.1 Ajonaikaisen arkkitehtuurin kuvaaminen**

Ajonaikaiseen arkkitehtuuriin kuvaamisessa keskitytään prosessinäkymään ja siihen kuuluu samanaikaisuusvaatimusten tunnistaminen, prosessien ja säikeiden tunnistaminen, prosessien elinkaarien tunnistaminen, prosessien muunta-

minen toteutukseen ja mallinnuselementtien sijoittelu prosessien kesken. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Samanaikaisuusvaatimuksilla määritellään, kuinka laajasti järjestelmä vaatii rinnakkaista suoritusta ja kuinka hyvin rinnakkaisuutta pystytään tukemaan. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Prosessien ja säikeiden tunnistamisessa luodaan jokaiselle järjestelmän toiminnanohjaukselle prosessi tai säie. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Prosessi tarjoaa raskaan sarjan toiminnanohjausta, se on erillinen yksikkönsä ja se voidaan jakaa säikeiksi. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Säie tarjoaa kevyen sarjan toiminnanohjausta ja se toimii ympäröivän prosessin yhteydessä. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Prosessien elinkaaren tunnistamisessa selvitetään, koska tunnistetut prosessit ja säikeet luodaan ja tuhoetaan. Yksiprosessiarkkitehtuurissa prosessit luodaan ohjelman käynnistyessä ja tuhoetaan ohjelman sammuessa. Moniprosessiarkkitehtuurissa uusia prosesseja tehdään ohjelman käynnistyessä tehdystä prosessista. Moniprosessiarkkitehtuurissa jokainen prosessi on tuhottava yksilöllisesti. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Prosesseja muunnettaessa toteutukseen on otettava huomioon prosessien kytkennät, suorituskyykyvaatimukset, järjestelmän prosessien ja säikeiden rajoitukset, olemassa olevat säikeet ja prosessit sekä prosessien välisen kommunikation resurssien saatavuus. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

Suunnitteluelementtien jakamisen tarkoituksena on määritellä, missä prosesseissa luokat ja alijärjestelmät tulisi olla suorituksessa. Jokainen luokka ja alijärjestelmä täytyy olla suorituksessa vähintään yhdessä prosessissa. (Ajonaikaisen arkkitehtuurin kuvaaminen 2009.)

## 6 Käytetyt tekniikat

Tässä luvussa käydään läpi työssä käytetyt tekniikat.

### 6.1 Microsoft .NET

.NET Framework on Microsoftin kehittämä Windowsille tarkoitettu sovelluskehys, joka on suunniteltu toimivaksi hajautetuissa ympäristöissä eli joissa tietokoneet ja laitteet ovat yhteydessä Internetin kautta. (Moghadampour 2011.)

Sovelluskehys on ohjelmisto, joka muodostaa rungon sen päälle rakennettaville ohjelmille. Sovelluskehysten tarkoitus on nopeuttaa uusien ohjelmistojen valmistamista toimimalla ohjelmoinnin apuvälineenä. Kehys onnistuu tavoitteessaan tarjoamalla valmiiksi rakennettuja osia, joita ei tarvitse kirjoittaa uudelleen ohjelman kehityksen aikana. (Moghadampour 2011.)

.NET ilmestyi ensimmäisen kerran vuonna 2000. Se tarjoaa suuren toiminnallisuuskirjaston sekä koodikielen vapauden eli melkein millä tahansa kielellä kirjoitettu koodi toimii myös toisella kielellä. (Moghadampour 2011.)

.NET-sovelluskehyselle kirjoitetut ohjelmat toimivat sovellusympäristössä nimeltään Common Language Runtime (CLR). CLR on virtuaalikone, joka tarjoaa turvallisuus-, muisti- ja poikkeuksenhallintapalveluita. Kirjasto sekä CLR yhdessä ovat .NET Framework. (Moghadampour 2011.)

Ohjelmat tehdään .NET-sovelluskehyselle yhdistämällä oma kirjoitettu koodi sekä .NET ja kirjastot keskenään. Useimmat Windows-ohjelmat käyttävät .NET-sovelluskehystä. Visual Studio on .NET-sovelluksien kehittämiseen tehty työkalu. (Moghadampour 2011.)

#### 6.1.1 Ominaisuudet

**Yhteensopivuus** eli uuden ja vanhan toimiminen samassa ympäristössä. .NET-sovelluskehys tarjoaa tavan päästä toiminnallisuuteen käsiksi ohjelmissa, jotka suoritetaan .NET-kehysten ulkopuolella. (Microsoft. .NET.)

**Kieliriippumattomuus** eli .NET-kehysten tarjoama Common Type System (CTS). CTS määrittelee kaikki mahdolliset tietotyypit sekä ohjelmointirakenteet, joita CLR tukee ja myös, kuinka ne toimivat keskenään noudattaen Common

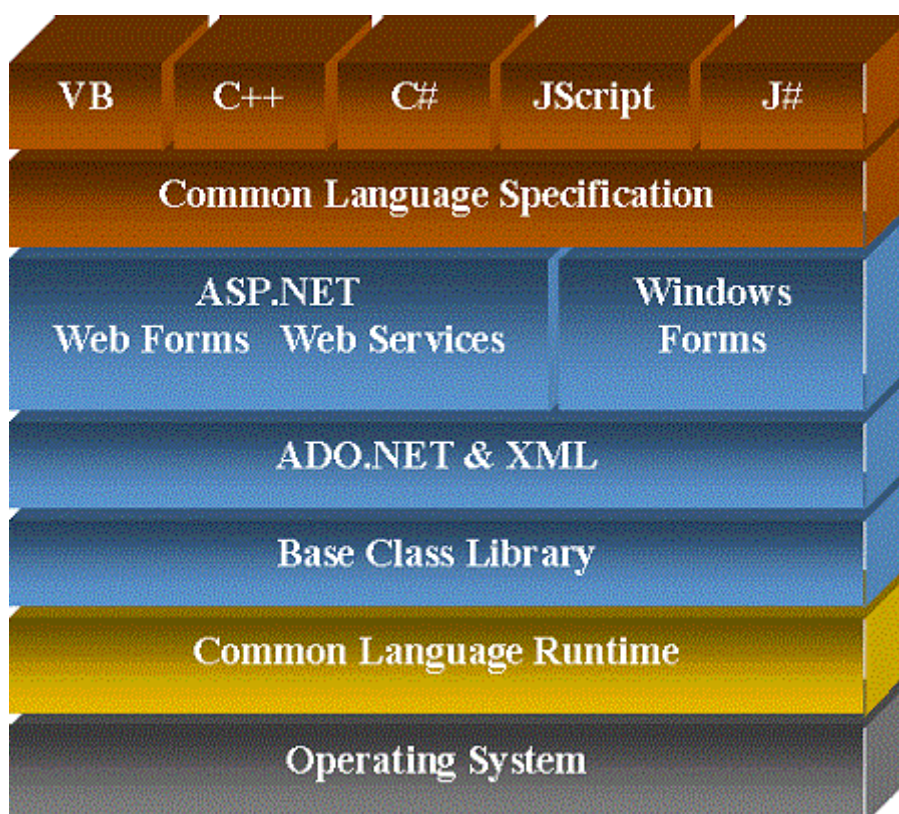
Language Infrastructure (CLI) määrittämiä. CTS auttaa .NET-yhteensopivilla ohjelmointikielillä kirjoitettujen ohjelmien välistä tiedonsiirtoa. (Microsoft. .NET.)

**Perusluokkakirjasto** (Base Class Library, BCL) on osa kehysluokkakirjastoa (Framework Class Library, FCL). BCL on toiminnallisuuskirjasto kaikille .NET-ohjelmointikielille. BCL tarjoaa luokkia, jotka suorittavat perustoimitoja, kuten tiedoston luku ja kirjoitus ja graafinen piirto. BCL sisältää CLR:n tarvitsemat perusluokat. (Microsoft. .NET.)

**Siirrettävyys**, vaikka Microsoft ei olekaan toteuttanut .NET-kehystä muille, kuin Windowsille. .NET:lle on kuitenkin tehty kehityspohja muille alustoille. (Microsoft. .NET.)

### 6.1.2 Arkkitehtuuri

.NET koostuu useasta eri komponentista (Kuva 10)



Kuva 10. .NET arkkitehtuuri (Microsoft. .NET-kuva)

Kuva 10 esittää .NET arkkitehtuuria ja sen osia.

Luokkakirjastoja on kaksi: perusluokkakirjasto, Base Class Library (BCL) ja kehysluokkakirjasto, Framework Class Library (FCL). BCL sisältää tärkeimmät CLR:n vaatimat toiminnot, kun taas FCL on BCL:n ylempi luokkakirjasto, joka sisältää kaikki .NET-sovelluskehityksen mukana tulevat luokkakirjastot. (Microsoft. .NET.)

CLR mahdollistaa dynaamisen muistinhallinnan, joten käyttäjän ei itse tarvitse luoda mitään koodia dynaamisen muistin varaamista ja vapauttamista varten. CLR tunnistaa, koska objekti on käytössä, tarkistamalla onko missään käynnissä olevassa ohjelmassa viittauksia siihen. Jos ei ole, objektin voi sammuttaa ja poistaa ja sen käyttämän muistin vapauttaa. (Microsoft. .NET.)

.NET-sovelluskehys sisältää oman roskienkeruuhjelman. Se alkaa toimia vasta, kun tietty muistimäärä on käytössä. Roskienkeruu tapahtuu tietyin väliajoin. Ajaessaan tarkistusta läpi, roskienkeruu pysäyttää ohjelman ja merkitsee jokaisen objektin, joka on käytössä. Jos jokin ei ole käytössä, on se valmis kerättäväksi. Tämän jälkeen ohjelma tarvitsee pakata kompaktiksi uudestaan eli poistaa tyhjät muistivälit juuri poistettujen objektien kohdalta. Ohjelman suoritus jatkuu keräyksen valmistuttua. Jokaisella kierroksella objekteille merkitään sukupolvi. Uusimmat objektit merkitään 0-tason sukupolvelle. Jos selviää ensimmäisestä keräyksestä, merkitään 1-tason sukupolvelle. Jos selviää vielä kerran keräyksestä, merkitään 2-tason sukupolvelle. Tämä auttaa parantamaan roskienkeruun tehokkuutta, koska ylemmän sukupolven objekteja ei tarvitse kerätä niin usein. (Microsoft. .NET.)

## 6.2 C#-ohjelmointikieli

C# on Microsoftin kehittämä ohjelmointikieli .NET-ympäristöön. Sen suunnittelussa tähdättiin moniin eri asioihin:

- C# tarkoitettiin yksinkertaiseksi, moderniksi, yleishyödylliseksi olio-ohjelmointikieleksi.
- Sen kuului tukea kirjoituksen, rajojen sekä käyttämättömien muuttujien tarkistusta ja hallita muistinkäyttöä käyttämättömien objektien osalta.
- C# tarkoitettiin käytettäväksi keskenään verkossa olevien koneiden sovellusten kehittämiseen.

- Ohjelmoijan siirtyminen sekä lähdekoodin siirrettävyys oli tärkeä osa kieltä kehitettäessä kuten myös kansainvälistymisen mahdollisuus.
- Kielellä voi kirjoittaa ylläpidettyjen ja upotettujen, niin pienten kuin hyvin suurtenkin, järjestelmien sovelluksia.
- C# tehovaatimukset kehitettiin pieniksi.

C# luotiin vuonna 2000, samoihin aikoihin kuin .NET alkujaan julkistettiin. Se perustui muun muassa C++-, Java- ja Delphi-kieliin. Alun perin C#:in sanottiin muistuttavan paljon Javaa, mutta nykyään yhtäläisyys Javaan on pienentynyt molempien kielten kehittyessä. Uusin versio on C# 5.0. (Microsoft. C#.)

### **6.3 Relaatiotietokanta**

Tässä luvussa käydään läpi erilaisia relaatiotietokantoja, miten niihin luodaan yhteys ja miten kantoihin tehdään kyselyjä ja muita lausekkeitä. Relaatiotietokanta on kokoelma tietoja ja niiden välille voidaan luoda yhteyksiä. Relaatiotietokannat luotiin suurien tietomäärien käsittelyjen helpottamiseksi.

#### **6.3.1 Microsoft Access**

Access on Microsoft Office -ohjelmapakettiin kuuluva relaatiotietokantojen käsittelyohjelma, joten se toimii hyvin yhteen muiden Microsoft Office paketin ohjelmien, kuten Microsoft Excelin kanssa. Access tarjoaa graafisen käyttöliittymän lausekkeiden ja taulujen tekoon, jolloin käyttäjän ei tarvitse osata SQL:aa juuri ollenkaan. Access tukee 255 yhtäaikaista käyttäjää. Se soveltuu myös vain pienten ja keskisuurten aineistojen käsittelyyn, koska tietokannan koko on rajoitettu 2 gigatavuun. Access skaalautuu suuremmaksi linkittämällä useita Access-tietokantoja yhteen tai käyttämällä Microsoft SQL Serveriä. Näin menetellen tietojen ja käyttäjien määrä voi kasvaa suuremmaksi. Sitä voidaan myös käyttää käyttöliittymänä SQL-tietokantoihin. (Microsoft. Access.)

#### **6.3.2 MySQL**

MySQL on relaatiotietokantaohjelmisto, joka on nykyisin Oraclen omistuksessa. MySQL-tietokannan yhteyteen rakennettava ohjelman logiikka koodataan usein PHP-, Python- tai Perl-ohjelmointikielillä ja itse sivut julkaistaan Apache-



webpalvelimella. MySQL sisältää myös rajapinnan eri ohjelmointikielille kuten C:lle, C++:lle, C#:lle, Javalle, Smalltalkille, Rubyille ja TCL:lle. (Oracle. MySQL.)

### 6.3.3 Tietokantayhteydet

Tietokantayhteys on yhteys, jolla päästään käsiksi ohjelmasta tietokannan sisältöön. Tässä luvussa käydään läpi kaksi vaihtoehtoista menetelmää tietokantayhteyden muodostamiseen.

**Dataset** on Visual Studiosta valmiiksi löytyvä toiminto, jolla voi ottaa helposti tietokannan taulut ohjelman käyttöön. Jokaiselle taululle täytyy luoda oma Dataset, jotta sitä voi käyttää. Kun taulut on lisätty Datasetsiksi, pystytään kaikki lisätyt taulut ja niiden kentät sekä kenttien tyypit näkemään suoraan Visual Studiossa.

Datasettia käyttäessä luodaan kyselyt ohjelman käyttöön tuotuihin tauluihin, jonka jälkeen niitä voi kutsua kyseisen taulun Datasetin kautta. Tietokannan tiedot ovat, kuitenkin vain tietokannassa. Datasetin kautta tietokannan tietojen muuttaminen, poistaminen ja lisääminen onnistuvat erittäin helposti. Dataset auttaa kyselyjen luomisessa, mutta ne voi kirjoittaa myös kokonaan itse.

**MySQL connection string** on MySQL-tietokantoja varten kehitetty yksinkertainen muuttuja, jolla saadaan yhteys MySQL-tietokantaan.

Jotta MySQL connection stringiä voisi käyttää, on tietokoneeseen asennettava MySQL Connector-komponentti. Tämä mahdollistaa MySQL-luokkakirjaston käytön Visual Studiassa. MySQL Connectorin asennuksen jälkeen voidaan tietokantayhteys luoda kirjoittamalla yhteysrivi MySqlConnection-luokan käyttöön.

Alla on esimerkki yhteyden avaamisesta ja sulkemisesta MySQL connection stringiä käyttämällä.

Ensin luodaan itse yhteysrivi ja yhteysmuuttuja, joka on tyypiltään MySqlConnection. Riville kirjoitetaan mihin osoitteeseen otetaan yhteyttä, tietokannan nimi, käyttäjätunnus ja salasana, jos salasanaa tarvitaan. Tässä tapauksessa salasanaa ei ole käytössä.

```
string yhteys = "Server=localhost;Database=kataso;Uid=root;"
```

```
MySqlConnection sqlcon;
```

Kun yhteyden tiedot on määritetty yhteysriviin ja yhteysmuuttuja sqlcon luotu, voidaan yhteys luoda tietokantaan alla olevan esimerkin mukaan.

```
sqlcon = new MySqlConnection(yhteys);
```

Tämän jälkeen yhteys voidaan avata Open-komennolla.

```
sqlcon.Open();
```

Kun yhteyttä ei enää tarvita, kannattaa se sulkea Close-komennolla.

```
sqlcon.Close();
```

Open- ja Close-metodit ovat MySqlConnection-luokan valmiita metodeja ja sillä on myös useita muita valmiita metodeja. (Oracle. MySQL.)

## 6.4 SQL-kieli

Relaatiotietokantoihin kyselyjä tehdessä käytetään usein IBM:n kehittämää SQL-kyselykieltä. SQL-kielessä on verrattain pieni määrä peruskomentoja, mutta niitä yhdistelemällä pystytään luomaan erittäin monimutkaisia lausekkeita. Peruskomentoja ovat SELECT, jolla voidaan valita määritetty tieto tietokannasta. INSERT INTO, jonka avulla voidaan lisätä tietokannan tauluihin uusia rivejä. UPDATE, jolla voidaan vaihtaa jo löytyvien rivien arvoja. DELETE, jonka avulla pystymme poistamaan tietokannasta määriteltyjä rivejä. Kyselyjen tarkentamiseen löytyy myös useita eri komentoja kuten WHERE, AND ja OR. Tyhjätilymerkkejä ei huomioida SQL-lausekkeita suoritettaessa. (W3Schools. SQL.)

SQL voidaan jakaa eri kielielementteihin, joista esimerkkikuva alla (Kuva 11).



Kuva 11. SQL-elementit (Wikipedia. SQL-elementit)

Lauseke, joka on olennainen osa lausetta tai kyselyä. Lauseke alkaa aina jollain komennolla. Kuvassa Kuva 11 on esimerkki SQL-lauseesta, jossa on kolme eri lauseketta: UPDATE, SET ja WHERE. (W3Schools. SQL.)

Määritelmällä voidaan tuottaa suureita tai tauluja. Kuvassa Kuva 11 määritelmä on `DateTime.Today`. Määritelmän täytyy olla taulun kenttää vastaavassa muodossa. (W3Schools. SQL.)

Predikaatti määrittää ehdot, joilla voidaan rajoittaa lauseita tai kyselyitä tai muuttaa ohjelman kulkua. Esimerkiksi kuvassa Kuva 11 `paiva`-kentän arvoa verrataan `DateTime.Today` -arvoon, joka on kyseisen lauseen predikaatti. (W3Schools. SQL.)

Lause on lausekkeista muodostuva kokonaisuus, jossa suoritetaan esim. kysely eri kriteereillä. Lause päättyy puolipisteeseen. (W3Schools. SQL.)

Kysely, joka noutaa kriteerien mukaisen datan. Kysely alkaa aina `SELECT`-komennolla ja se on tärkeä osa SQL:a. (W3Schools. SQL.)

#### 6.4.1 Esimerkki tietokantakyselyn käytöstä

Ensin luodaan itse kysely, jossa haetaan tietyt tiedot tietyistä taulusta tietyillä ehdoilla.

```
kysely = "SELECT merkintatyyppiID, merkinta FROM merkintatyyppi
WHERE merkintatyyppi_tila = 1";
```

Seuraavaksi luodaan MySQL komento-muuttuja, jolla voidaan suorittaa kysely. Komennon ensimmäinen argumentti on kysely ja toinen yhteys.

```
MySqlCommand komento = new MySqlCommand(kysely, Muuttu-  
jat.sqlcon);
```

Tämän jälkeen luodaan MySqlConnection-tyyppinen lukija-muuttuja, johon voi väliaikaisesti tallentaa kyselyn tulokset. Itse kysely suoritetaan ExecuteReader-komennolla.

```
MySqlDataReader lukija = new MySqlDataReader();
```

```
MySqlDataReader.lukija = komento.ExecuteReader();
```

Lukijasta voi nyt ottaa saadut tulokset talteen uuteen muuttujaan, joka tässä tapauksessa on tyyppiä DataTable. Siirtäminen onnistuu yksinkertaisella Load-komennolla. Tietojen talteen ottamiseen on olemassa myös muita muuttujatyyppejä.

```
DataTable merkintatyyppi = new DataTable();
```

```
merkintatyyppi.Load(lukija);
```

Näin ollen käytössä on tietokannasta kyselyn avulla haetut tiedot datataulun muodossa, josta niitä on helppo ja nopea käyttää.

## 7 Työvälineet

Tässä luvussa käydään läpi opinnäytetyössä käytetyt työvälineet.

### 7.1 Microsoft Visual Studio 2010

Visual Studio on Microsoftin ohjelmankehitysympäristö, jossa voidaan käyttää useita eri ohjelmointikieliä, kuten C#, C++ ja J#. Visual Studiota käytetään konsoli ja graafisten käyttöliittymien sekä Windows Forms ja Windows Presentation Foundation ohjelmien tekoon. Sillä voi myös ohjelmoida web-sivuja, -ohjelmia ja -palveluita. (Microsoft. Visual Studio.)

Visual Studio tukee monia eri kieliä palveluiden kautta, jolloin ohjelman eri toiminnot myös tukevat niitä kieliä. Sisäänrakennettu kielituki löytyy C, C++, VB.NET, C# ja F# -ohjelmointikielille. Muille ohjelmointikielille löytyy tuki ladattavina palveluina, jotka asennetaan erikseen. Visual Studion voi myös ladata

kielikohtaisena, jolloin siitä löytyy vakiona tuki vain valitulle kielelle. (Microsoft. Visual Studio.)

### **7.1.1 Koodieditori**

Visual Studio sisältää koodieditorin, joka tukee syntaksien korostamista ja koodin täydentämistä käyttäen IntelliSenseä muuttujiin, funktioihin ja metodeihin sekä myös silmukoihin ja kyselyihin. IntelliSense toimii mukana tulevilla kielillä ja myös XML:lla ja osalle kielistä web-sovelluksia luodessa. Koodieditoria käytetään jokaisen tuetun kielen kanssa. Täydennysehdotukset tulevat ponnausikkunaan koodin päälle. Täydennysvaihtoehdoissa lukee myös lyhyt kuvaus sen toiminnosta. Ponnausikkunan voi myös asettaa hieman läpinäkyväksi uusimmissa versioissa, jolloin koodin näkee alta. (Microsoft. Visual Studio.)

Koodieditorilla voi myös asettaa kirjanmerkkejä koodiin nopeaa navigointia varten. Muita hyödyllisiä navigointiapuja ovat suljettavat koodialueet sekä sanahaku jne. Koodieditorissa on monen asian leikepöytä ja työlista. Se myös tukee valmiita koodipohjia, jotka ovat tallennettuja malleja toistuvasta koodista. Ne voidaan lisätä koodiin ja muokata tarpeen mukaan. Mallien muokkaustyökalu on sisäänrakennettu. Yllämainitut työkalut ovat leijuvia ikkunoita, jotka voidaan asettaa piiloutumaan automaattisesti silloin, kun niitä ei käytetä tai asettaa ne ruudun reunaan. (Microsoft. Visual Studio.)

Koodieditori tukee myös koodin siistimistä, esimerkiksi parametrien uudelleenjärjestäminen, muuttujien tai metodien uudelleennimeäminen. (Microsoft. Visual Studio.)

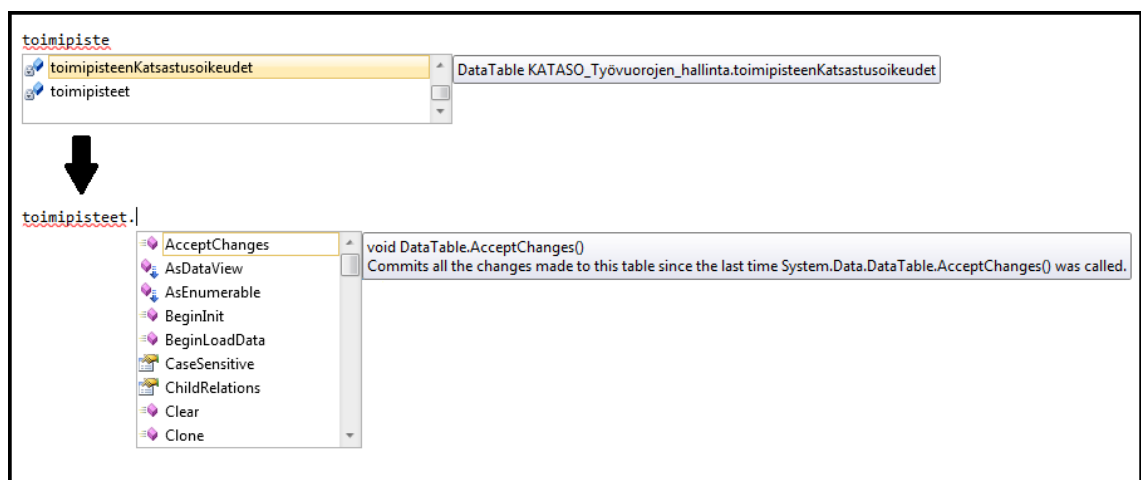
Koodieditorin todella hyvä ominaisuus on myös taustakääntäminen. Koodia kirjoitettaessa Visual Studio kääntää sitä jatkuvasti, jolloin se voi huomauttaa mahdollisista virheistä heti. Tämä ominaisuus on tuettu kaikilla mukana tulevilla kielillä. (Microsoft. Visual Studio.)

### **7.1.2 IntelliSense**

IntelliSense on Microsoftin kehittämä älykäs koodintäydennys-työkalu. Se ilmestyi jo ensimmäisen kerran 1996 Visual Basicin mukana, joka toimi sen pääasiallisena kehityspohjana. Nykyään IntelliSense tulee Visual Studion mukana ja on

tuettu sen mukana tulevilla kielillä. IntelliSense aktivoituu heti, kun kirjoittamisen aloittaa. IntelliSenseä käytettäessä se osaa valita suotuisimman vaihtoehdon vakioehdokkaaksi täydennyslistasta jo ensimmäisen kerran jälkeen. IntelliSensen voi myös pakottaa näyttämään koko listansa painamalla Visual Studion pikanäppäintä Ctrl + J tai Ctrl + Space. (Microsoft. Visual Studio.)

IntelliSense on nopeuttanut koodin kirjoittamista, koska ei tarvitse muistaa niin montaa komentoa ja täydentämisellä säästyy kirjainten näppäilyltä. Täydentäminen tapahtuu painamalla Tab- tai Enter-painiketta valitun vaihtoehdon kohdalla tai laittamalla kielikohtaisen merkin. Esimerkki IntelliSensen toiminnasta näkyy kuvassa Kuva 12. (Microsoft. Visual Studio.)



Kuva 12. IntelliSensen toiminta

Yllä olevassa kuvassa Kuva 12 nähdään, että IntelliSense etsii kaikki mahdolliset eri vaihtoehdot siihen mennessä kirjoitetun tekstin perusteella. Kun valinta on tehty, saadaan valitulle toiminnolle, muuttujalle ynnä muille sellaisille esiin sille kuuluvat toiminnot kirjoittamalla piste tai jokin muu tarvittava välimerkki sen perään. (Microsoft. Visual Studio.)

### 7.1.3 Testaustyökalu

Visual Studiossa tulee mukana testaustyökalu, joka toimii suoritettavalla koodilla tai konekielellä. Sitä käytetään Visual Studiolla luotujen ohjelmien testaukseen ja voidaan myös liittää käynnissä oleviin prosesseihin tarkkailemaan ja etsimään ja korjaamaan virheet. Jos lähdekoodi on saatavilla, osaa testaustyökalu näyttää koodin sitä suoritettaessa. Jos lähdekoodia ei ole saatavilla, osaa

työkalu silti purkaa suoritettavat toiminnot ihmisystävälliseksi tekstiksi. Työkalu tukee monisäieohjelmia ja se voidaan asettaa käynnistymään, kun jokin ohjelma Visual Studion ulkopuolella lakkaa toimimasta. (Microsoft. Visual Studio.)

Testaustyökalulla voi asettaa katkaisupisteitä, jotka pysäyttävät ohjelman suorituksen siinä pisteessä väliaikaisesti, ja tarkkailukohtia, jotka tarkkailevat muuttujien arvoja suorituksen edetessä. Katkaisupisteet voivat olla ehdollisia, jolloin ne pysäyttävät vain, kun ehto toteutuu. Koodia voi edetä askel kerrallaan. Työkalu tukee Muokkaa ja jatka -ominaisuutta, joka mahdollistaa koodin muokkauksen testauksen yhteydessä (toimii vain 32-bittisellä versiolla). Testauksen aikana muuttujan arvoja voi tarkastella viemällä hiiren cursorin muuttujan päälle ja siitä myös tarvittaessa muokata. (Microsoft. Visual Studio.)

## **7.2 XAMPP**

XAMPP on ilmainen avoimen lähdekoodin ohjelma, joka on suunniteltu paikalliseksi web-serveriksi, jolla voi testata erilaisia web-sovelluksia omalla koneella. Tämä tarkoittaa sitä, että Internet-yhteyttä ei välttämättä tarvita tietokantayhteyden luomiseen ja yhteys on myös erittäin nopea sen toimiessa paikallisesti. (XAMPP.)

XAMPP tarkoittaa

- X - Cross-platform eli toimii monilla eri alustoilla
- A - Apache HTTP Server eli web-serveri
- M - MySQL eli relaatiotietokanta
- P - PHP eli ohjelmointikieli, jota usein käytetään web-sovelluksissa
- P - Perl eli ohjelmointikieli.

(XAMPP.)

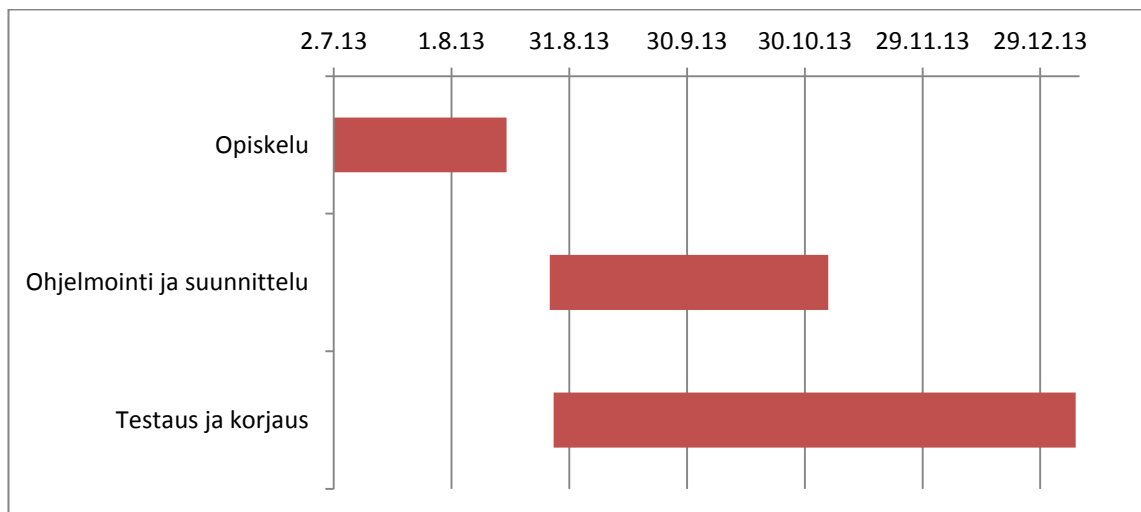
## 8 Opinnäytetyöprojektin toteutus

Tässä luvussa käydään läpi opinnäytetyöprojektin suunnittelu, vaiheet ja tehtävien työmäärät.

Projektin suunnittelu ja toteutus noudatteli XP-menetelmää. Käytännössä emme noudattaneet sitä täydellisesti ja joitain osia menetelmästä jäi käyttämättä, kuten testivetoinen kehitys ja iteraatioiden pituus, joka meillä vaihteli suuresti.

### 8.1 Projektin suunnitteluvaiheet ja aikataulu

Jaoimme projektin karkeasti kolmeen eri vaiheeseen: opiskelu, suunnittelu ja ohjelmointi sekä ohjelman testaaminen ja korjaaminen (Kuva 13).



Kuva 13. Gantt-kaavio projektin vaiheista

Kuvassa Kuva 13 on projektin vaiheet. Opiskelimme tarvittavat tekniikat heinä-elokuussa 2013, mutta ohjelmoinnin lomassa jouduimme kuitenkin käytännössä opiskelemaan lisää. Ohjelmointi ja suunnittelu tapahtuivat elokuun lopusta lokakuun alkuun. Ohjelman testaaminen ja korjaaminen sijoituivat syyskuun alusta tammikuun alkuun.

#### 8.1.1 Opiskelu

Projekti lähti liikkeelle opiskelemalla eri tekniikoita, joita odotimme tarvitsevamme ohjelman koodaamisessa. Opiskelun jälkeen pidimme asiakastapaamisen, joita meillä oli projektin alkuvaiheessa melkein joka viikko. Tapaamisissa selvitimme mahdollisimman tarkasti, mitä asiakas ohjelmalta haluaa, ja kävimme läpi



kaikki epäselvyydet määrittelyn suhteen. Loppuvaiheessa pidimme palaveria harvemmin; vain silloin kuin ilmeni uusia epäselvyyksiä tai, kun piti päättää, mitä tehdään seuraavaksi. Kirjoitimme aina muistiota jokaisesta pidetystä palaverista. Muistioiden ja muiden dokumenttien ja tiedostojen jakamiseen käytimme Google Drive -tiedostonjakopalvelua.

### **8.1.2 Suunnittelu ja ohjelmointi**

Aloitimme suunnittelun hahmottamalla lomakkeiden ulkomuodot, jonka jälkeen otimme niistä käsittelyyn yhden kerrallaan ja suunnittelimme tarkemmin lomakkeen ulkomuodon, toiminnot ja niiden toteutuksen. Jotkin lomakkeista olivat niin tiiviisti liitoksissa toisiinsa, että työstimme niitä samanaikaisesti pystyäksemme pitämään niiden välisen yhteyden mahdollisimman toimivana. Toisinaan aloitimme seuraavan lomakkeen suunnittelun jo ennen kuin edellisen lomakkeen koodaaminen oli valmis. Teimme näin silloin, kun jäimme jumiin koodaamiseen suhteen tai kun huomasimme, että jokin asia täytyy käydä läpi asiakkaan kanssa. Pääasiassa kuitenkin pyrimme etenemään suunnittelemassamme järjestyksessä.

Ohjelman toiminnallista määrittelyä käytiin uudelleen läpi palaverissa ja siihen tuli melko paljon muutoksia. Ohjelmoinnin suunnittelu oli lähestulkoon täysin meidän hallinnassa ja toiminnallisuuteen tuli hyvin paljon muutoksia alkuperäiseen määrittelyyn verrattuna. Etenkin ohjelmoinnin alkuvaiheessa jouduimme usein palaamaan suunnittelupöydän ääreen joidenkin toimintojen toteuttamisen suhteen, kun ne eivät toimineet niin kuin niiden oli tarkoitus tai, jos löysimme uuden paremman keinon toiminnon toteuttamiseen.

Laitteistoarkkitehtuurin toteuttamiseen on kaksi eri suunnitelmaa:

Ensimmäinen suunnitelma on, että tietokanta sijoitetaan Lappeenrannan A-Katsastuksen paikalliselle palvelimelle, jonka kautta sitä voidaan käyttää kaikilla Lappeenrannan yksikön PC-työpisteillä ja Internet-yhteys ei ole pakollinen.

Toinen suunnitelma on, että tietokanta sijoitetaan A-Katsastuksen pääkonttorin palvelimelle Helsinkiin, jonka kautta sitä voidaan käyttää Lappeenrannan yksikön kaikilla PC-työpisteillä. Tässä suunnitelmassa Internet-yhteys on pakollinen.

Vielä ei ole varmaa kummalla suunnitelmalla ohjelmaa aletaan lopulta käyttää.

Ohjelmoimme ja suunnittelimme ohjelman toimintoja palaverien välillä niin, että jätimme työtuntien generoinnin viimeiseksi, koska se oli selvästi kaikista haastavin tehtävä. Työvuorojen generoinnin suunnittelua varten pidimme erikseen tapaamisen, jossa teimme toimintasuunnitelman sen toteuttamista varten.

Tietokantavaihtoehtoja meillä oli kaksi: Microsoft Access ja MySQL. Määrittelyssä tietokannaksi oli ehdotettu Access, mutta päädyimme käyttämään pienen alkutestauksen jälkeen MySQL-tietokantaa. Molemmat olivat tuttuja jollain tasolla, mutta MySQL tuntui kätevämmältä sekä helpommin muokattavalta.

Tietokanta oli sijoitettu paikalliselle XAMPP-palvelimelle. Meillä oli käytössä XAMPP-Lite, joka on XAMPPin kevennetty versio.

Tietokantayhteyden muodostamiseen meillä oli kaksi esiteltyä vaihtoehtoa. Käytimme ensin Dataset-yhteyttä, mutta vastaan tuli pieniä ongelmia joidenkin kyselyjen suorittamisessa ja muuttuvan tietokantarakenteen kanssa, joten päädyimme käyttämään MySQL connection stringiä.

Pysyimme ohjelmoinnin suhteen hyvin aikataulun mukana ja saimme toimitettua testiversion ohjelmasta asiakkaalle aikataulun mukaisesti.

### **8.1.3 Ohjelman testaaminen ja korjaaminen**

Ohjelman testaamista teimme hieman aina uusien toimintojen yhteydessä, mutta suurin osa ohjelman testaamisesta tapahtui, kun ohjelman kaikki toiminnot olivat valmiita. Löysimme useita uusia virheitä ja testaaminen osoittautui erittäin kannattavaksi. Ohjelma oli myös testattavana testauskurssilla, jossa ohjelmasta löytyi useita virheitä.

Kun olimme saaneet ensimmäisen version ohjelmasta tehtyä, annoimme sen testattavaksi A-Katsastukselle ja aloimme kirjoittaa itse opinnäytetyöraporttia.

## **8.2 Tehtävien työmäärät**

Opiskeluun kului noin 50 tuntia ja se suoritettiin kesäloman aikana. Suunnitteluun ja ohjelmointiin käytimme noin 300 tuntia ja siihen sisältyi myös osittain

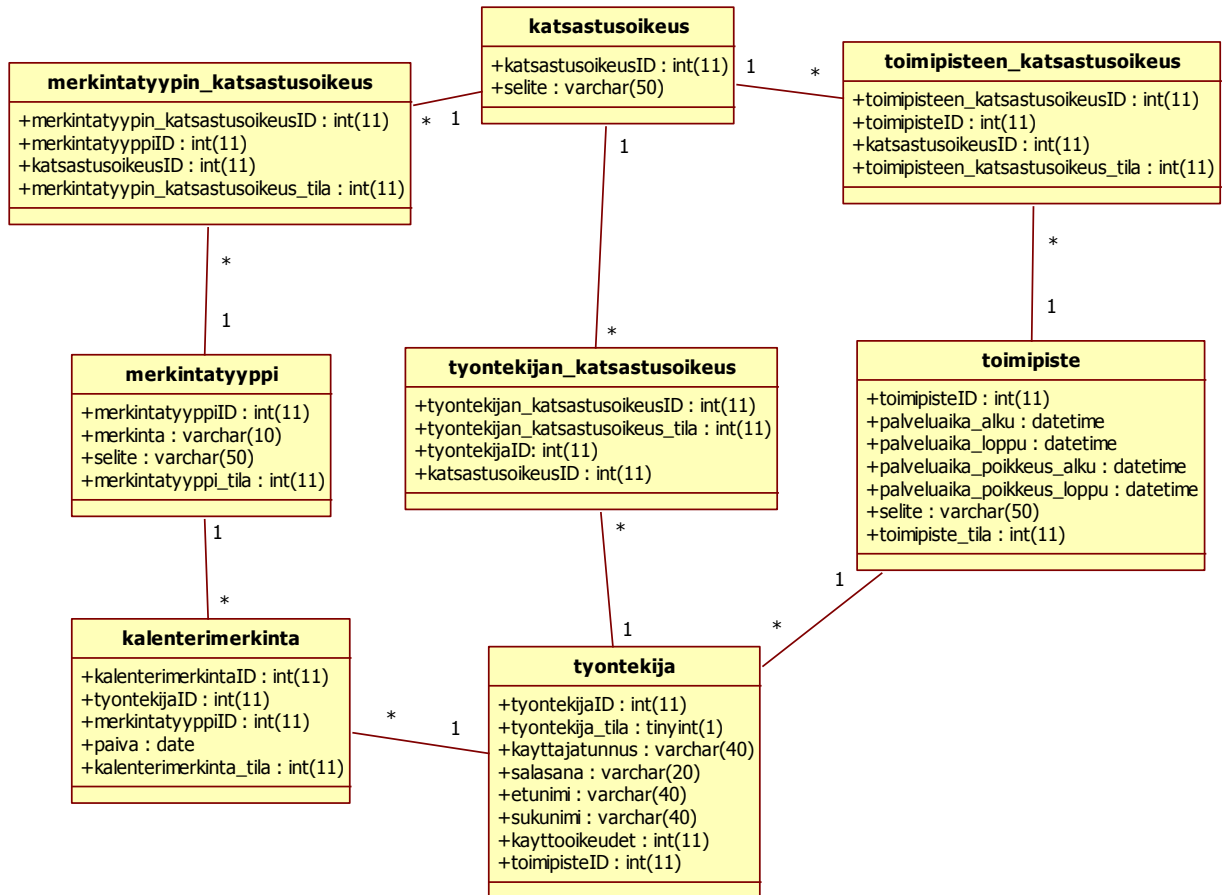
opiskelua. Ohjelman testaamiseen ja korjaamiseen käytimme noin 150 tuntia, koska testaamista tapahtui myös suunnittelu- ja ohjelmointivaiheessa.

## 9 Lopputuotteen esittely

Tässä luvussa käydään läpi valmiin lopputuotteen ulkomuoto ja toiminnot. Ohjelman toimintoja ovat työvuorojen tarkastelu ja tulostus, työvuorojenhallinta, työntekijöiden hallinta, merkintätyyppien hallinta ja toimipaikkojenhallinta.

### 9.1 Ohjelman tietokantarakenne

Teimme ohjelman tietokantarakenteeseen (Kuva 14) joitakin muutoksia määrittelyn alkuperäiseen tietokantarakenteeseen verrattuna.

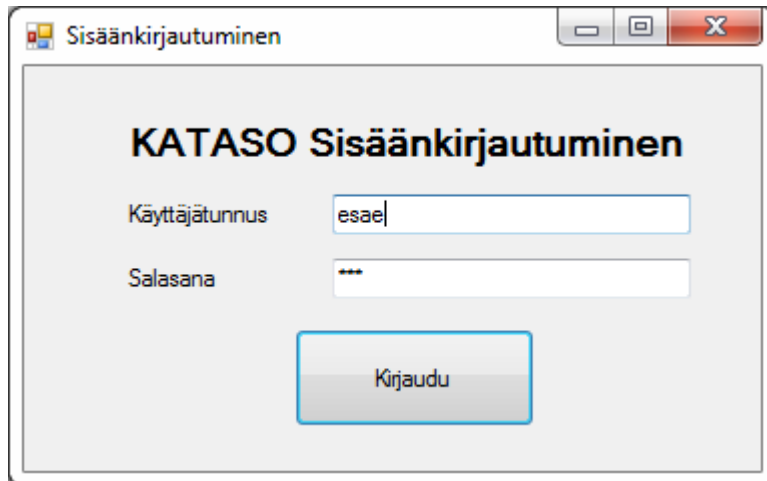


Kuva 14. Tietokantarakenne

Lisäsimme tietokantarakenteeseen (Kuva 14) kaksi uutta taulua merkintatyy-  
pin\_katsastusoikeus ja toimipisteen\_katsastusoikeus, joita alkuperäisessä mää-  
rittelyssä ei ollut. Muutimme myös osittain taulujen attribuutteja.

## 9.2 Sisäänkirjautuminen

Ohjelman käynnistyessä avautuu ensimmäiseksi kuvan Kuva 15 mukainen Si-  
säänkirjautuminen-lomake.

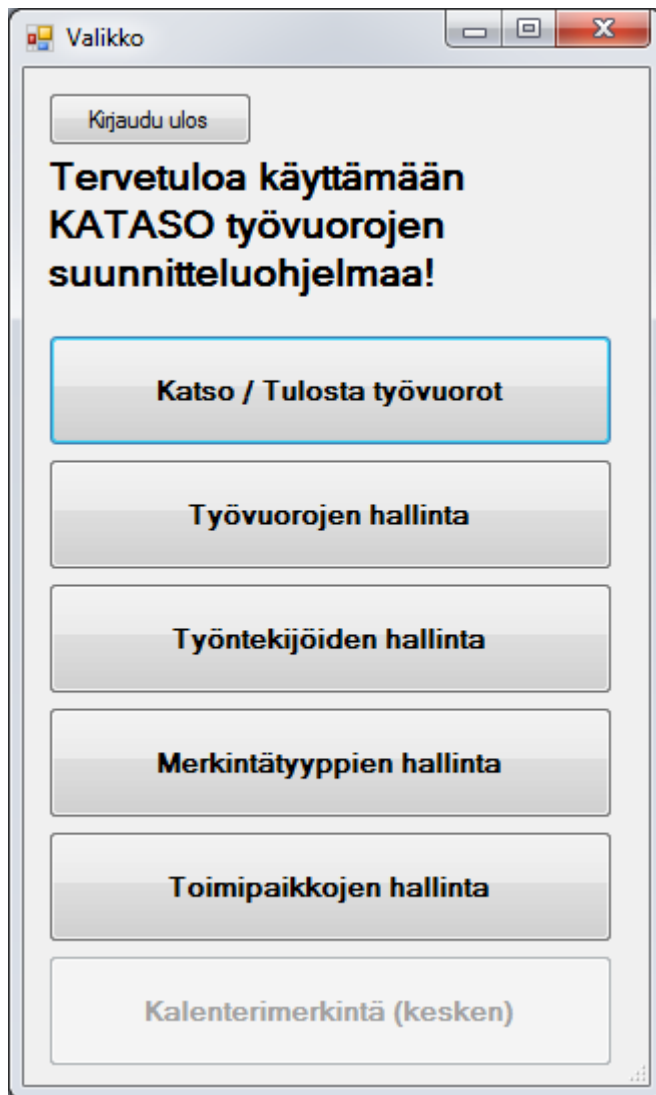


Kuva 15. Sisäänkirjautuminen

Sisäänkirjautuminen-lomakkeessa (Kuva 15) kirjoitetaan Käyttäjätunnus-  
kenttään käyttäjätunnus ja Salasana-kenttään käyttäjätunnusta vastaava sala-  
sana. Tunnukset on luokiteltu 3 eri luokkaan, jotka ovat pääkäyttäjä, esimies ja  
työntekijä. Tämän jälkeen voidaan kirjautua sisään painamalla Kirjaudu-  
painiketta. Ohjelma huomauttaa käyttäjää, jos salasana tai käyttäjätunnus kirjai-  
tettiin väärin.

## 9.3 Valikko

Jos käyttäjätunnus ja salasana kirjoitettiin oikein Sisäänkirjautuminen-  
lomakkeessa, avautuu käyttäjälle Valikko-lomake (Kuva 16).



Kuva 16. Valikko

Valikko-lomakkeesta (Kuva 16) on painike jokaiselle ohjelman toiminnolle sekä Kirjaudu ulos-painike, josta päästään takaisin Sisäänkirjautuminen-lomakkeelle. Jos käyttäjä kirjautuu sisään pääkäyttäjän oikeuksilla, pystyy hän käyttämään kaikkia toimintoja. Esimiehen oikeuksilla pystyy käyttämään kaikkia muita paitsi Toimipaikkojen hallinta -lomaketta. Työntekijän oikeuksilla voi avata vain Katso/Tulosta työvuorot -lomakkeen.

#### 9.4 Työvuorojen hallinta

Työvuorojen hallinta -lomake (Kuva 17) voidaan avata Valikko-lomakkeen (Kuva 16) painikkeista Katso / Tulosta työvuorot ja Työvuorojen hallinta.



manuaalisesti. Kun työvuorot on generoitu, täytyy ne tallentaa, jotta seuraava generointi ottaa huomioon juuri tehdyt työvuorot.

Tallenna-painike tallentaa kaikki taulukossa näkyvät merkinnät, jolloin kaikkien näkyvien merkintöjen väriksi vaihtuu vaalean keltainen.

Julkaise-painike julkaisee kaikki taulukossa näkyvät tallennetut työvuorot, jolloin työvuorojen taustaväri muuttuu valkoiseksi.

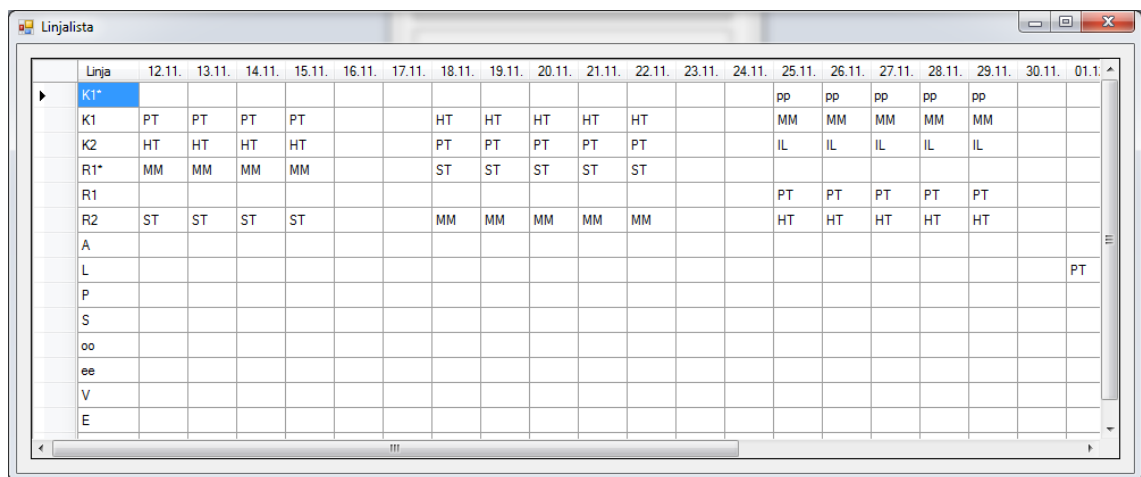
Peruuta-painike peruuttaa viimeksi tallennetut muutokset ja sitä voidaan painaa vain, jos on aikaisemmin painettu Julkaise- tai Tallenna-painiketta.

Tulosta-painikkeella voidaan tulostaa taulukossa näkyvät työvuorot.

Toimipiste valitaan ”Valitse toimipiste”-tekstin alla olevasta alavetovalikosta, jolloin tauluun tulostuu valitun toimipisteen työvuorolista.

Sulje-painike sulkee lomakkeen ja jos Linjalista on auki, se sulkeutuu myös.

Työvuorolista voidaan avata myös Valikon Työvuorojen hallinta -painikkeesta, jolloin työvuorolistan viereen avautuu Linjalista-lomake (Kuva 18).



The screenshot shows a window titled "Linjalista" containing a grid for shift scheduling. The grid has columns for dates from 12.11. to 01.12. and rows for different shift types and personnel. The first row is labeled "Linja" and contains shift codes (PT, HT, ST, MM, IL, PP) for each date. Subsequent rows are labeled K1\*, K2, R1\*, R2, A, L, P, S, oo, ee, V, and E, each containing corresponding shift codes for the same dates. A vertical scrollbar is visible on the right side of the grid.

Linja	12.11.	13.11.	14.11.	15.11.	16.11.	17.11.	18.11.	19.11.	20.11.	21.11.	22.11.	23.11.	24.11.	25.11.	26.11.	27.11.	28.11.	29.11.	30.11.	01.12.
K1*														pp	pp	pp	pp	pp		
K2	PT	PT	PT	PT			HT	HT	HT	HT	HT			MM	MM	MM	MM	MM		
R1*	MM	MM	MM	MM			ST	ST	ST	ST	ST			IL	IL	IL	IL	IL		
R1														PT	PT	PT	PT	PT		
R2	ST	ST	ST	ST			MM	MM	MM	MM	MM			HT	HT	HT	HT	HT		
A																				
L																				PT
P																				
S																				
oo																				
ee																				
V																				
E																				

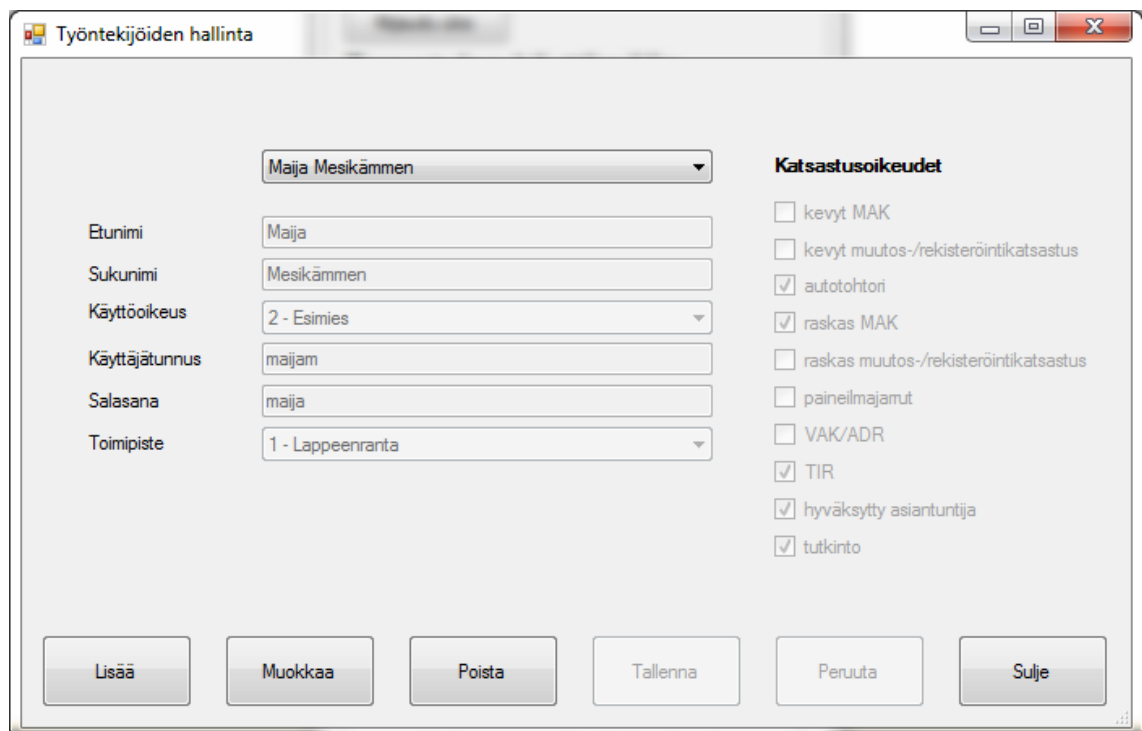
Kuva 18. Linjalista

Linjalista (Kuva 18) tulostaa näkyviin saman aikavälin, kuin samaan aikaan auki oleva työvuorolista. Linjalista näyttää onko kullakin päivällä jokaiselle työvuorolle jaettu työntekijä. Linjalistaan tulostuu myös muut kuin työvuoromerkinnät, jo-

ten siitä näkee myös onko kyseisillä päivillä jollain henkilöillä lomaa, koulutusta, ynnä muuta sellaista.

## 9.5 Työntekijöiden hallinta

Työntekijöiden hallinta -lomakkeessa (Kuva 19) voidaan muokata työntekijöiden ominaisuuksia, lisätä uusia työntekijöitä tai poistaa vanhoja työntekijöitä ja se voidaan avata Valikko-lomakkeen (Kuva 16) Työntekijöiden hallinta -painikkeesta.



Kuva 19. Työntekijöiden hallinta

Työntekijöiden hallinta -lomakkeessa (Kuva 19) voidaan valita haluttu työntekijä yläreunassa olevasta alasvetovalikosta, jolloin kenttiin tulostuu valitun työntekijän tiedot.

Lisää-painike tyhjentää ominaisuuskentät, jolloin niihin pystytään kirjoittamaan uuden työntekijän tiedot.

Muokkaa-painike mahdollistaa lomakkeen yläosassa olevasta alasvetovalikosta valitun työntekijän ominaisuuksien muokkaamisen.

Poista-painike poistaa valitun työntekijän.



Tallenna-painike tallentaa tehdyt muutokset.

Peruuta-painike peruuttaa edellisen tallennuksen.

Sulje-painike sulkee lomakkeen, jolloin palataan Valikko-lomakkeelle.

## 9.6 Merkintätyyppien hallinta

Merkintätyyppien hallinta -lomakkeessa (Kuva 20) voidaan muokata, poistaa tai luoda uusia merkintätyyppejä. Se voidaan avata Valikko-lomakkeen (Kuva 16) Merkintätyyppien hallinta -painikkeesta.

	Koodi	Selite
▶	K1*	7:45 - 16:00 Kevyt
	K1	8:00 - 16:15 Kevyt
	K2	9:00 - 17:15 Kevyt
	R1*	7:45 - 16:00 Raskas
	R1	8:00 - 16:15 Raskas
	R2	9:00 - 17:15 Raskas
	A	Arkipyhä
	L	Lomalla
	P	Poissa
	S	Sairauspoissaolo
	oo	Koulutus
	ee	Ylityövapaa
	V	Vapaa
	E	Ei enää käytettävissä
	Y	Ylityö
*		

Tallenna

Peruuta

Sulje

Kuva 20. Merkintätyyppien hallinta

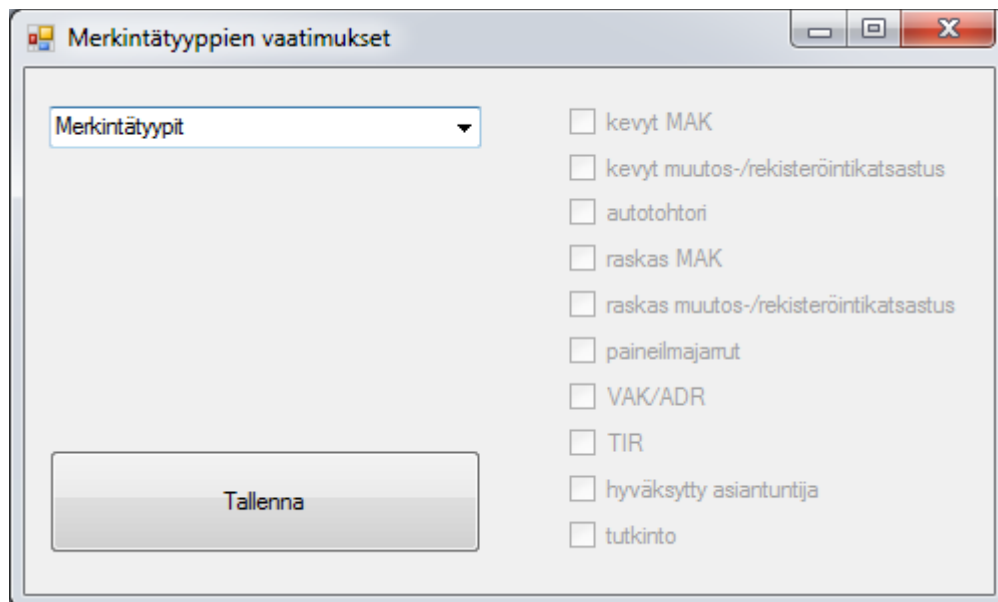
Merkintätyyppien hallinta -lomakkeessa (Kuva 20) merkintätyyppejä voidaan muokata suoraan tulostetusta listasta.

Uuden merkintätyyppin voi luoda kirjoittamalla Koodi ja Selite tyhjälle viimeiselle riville. Lomake luo uusia rivejä aina tarvittaessa.

Merkintätyypin voi poistaa pyyhkimällä kyseisen merkintätyyppin Koodi- ja Selite-sarakkeen tyhjäksi.

Sulje-painike sulkee Merkintätyyppien hallinta -lomakkeen sekä Merkintätyyppien vaatimukset -lomakkeen, jolloin palataan Valikko-lomakkeelle.

Merkintätyyppien vaatimukset -lomake (Kuva 21) avautuu Merkintätyyppien hallinta -lomakkeen viereen.



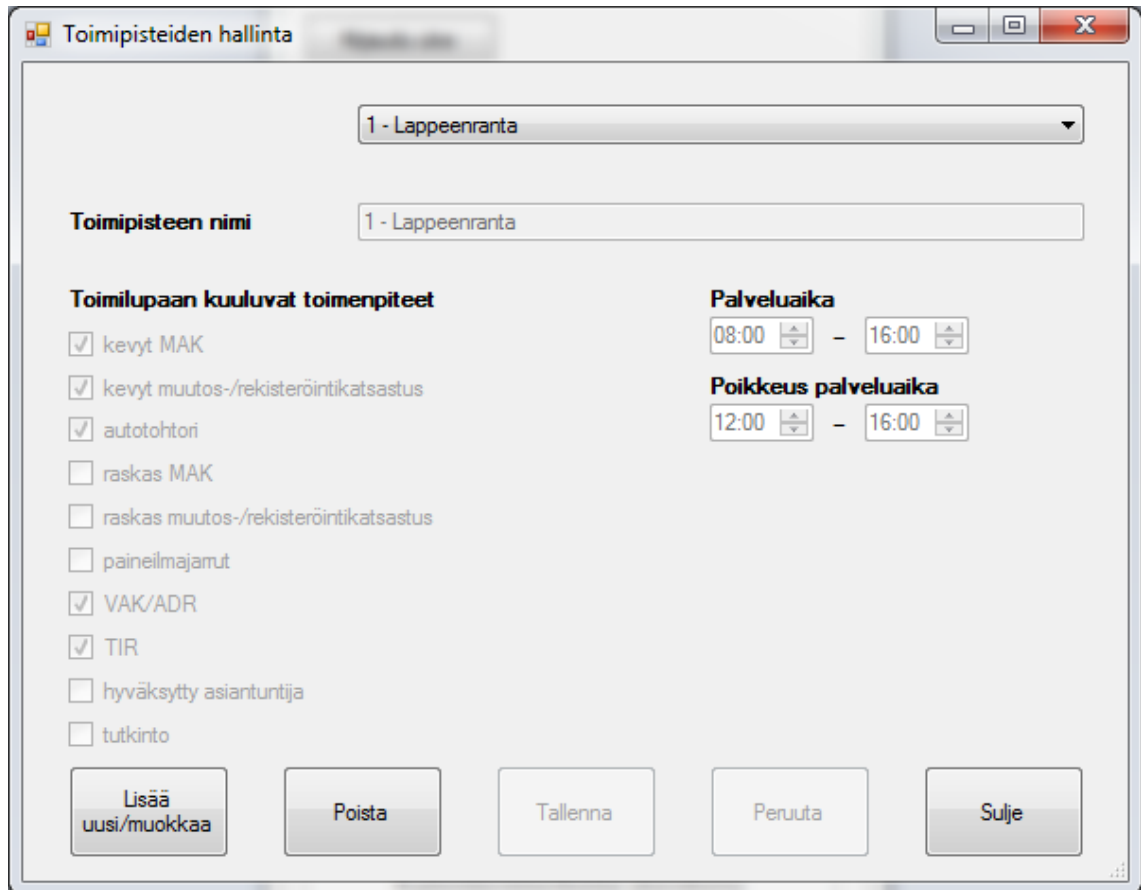
The image shows a software window titled "Merkintätyyppien vaatimukset". Inside the window, there is a dropdown menu on the left with the text "Merkintätyypit" and a downward arrow. To the right of the dropdown is a list of ten checkboxes, each followed by a label in Finnish. The labels are: "kevyt MAK", "kevyt muutos-/rekisteröintikatsastus", "autohtori", "raskas MAK", "raskas muutos-/rekisteröintikatsastus", "paineilmajamut", "VAK/ADR", "TIR", "hyväksytty asiantuntija", and "tutkinto". At the bottom left of the window is a button labeled "Tallenna". The window has standard Windows-style window controls (minimize, maximize, close) in the top right corner.

Kuva 21. Merkintätyyppien vaatimukset

Merkintätyyppien vaatimukset -lomakkeessa (Kuva 21) voidaan asettaa eri merkintätyypeille katsastusoikeusvaatimuksia. Valitsemalla alasvetovalikosta merkintätyyppin, tulevat sen oikeudet näkyviin valintaruutuihin. Jotta Merkintätyyppien hallinnassa uudet juuri luodut merkintätyypit tulisivat näkyviin Merkintätyyppien vaatimuksissa, täytyy ne ensin tallentaa. Tallenna-painike tallentaa valitulle merkintätyypille tehdyt muutokset.

## 9.7 Toimipisteiden hallinta

Toimipaikkojen hallinta -lomakkeessa (Kuva 22) voidaan muokata toimipisteiden ominaisuuksia, poistaa tai luoda uusia toimipisteitä ja se voidaan avata Valikko-lomakkeen (Kuva 16) Toimipisteiden hallinta -painikkeesta.



Kuva 22. Toimipisteiden hallinta

Muokatakseen jo luotua toimipistettä tai lisätäkseen uuden toimipisteen täytyy valita kuvan Kuva 22 yläreunassa olevasta alasvetovalikosta muokattava toimipiste tai "Lisää uusi" vaihtoehto. Tämän jälkeen painetaan Lisää uusi/muokkaa -painiketta, joka mahdollistaa valitun toimipisteen/uuden toimipisteen ominaisuuksien muokkaamisen.

Poista-painike poistaa valitun toimipisteen.

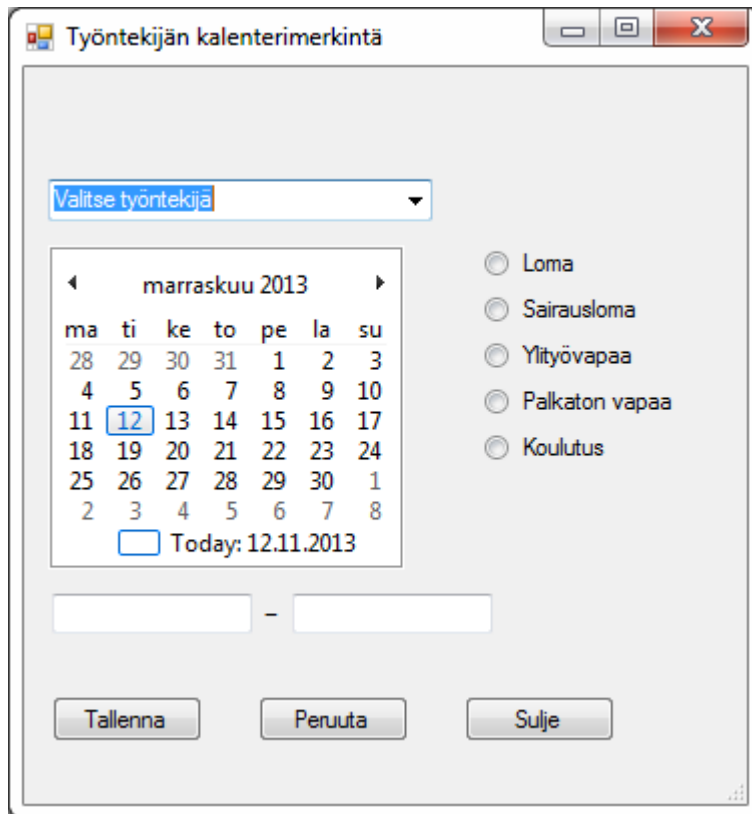
Tallenna-painike tallentaa tehdyt muutokset.

Peruuta-painike peruuttaa edellisen tallennuksen.

Sulje-painike sulkee lomakkeen, jolloin palataan Valikko-lomakkeelle.

## 9.8 Työntekijän kalenterimerkintä

Työntekijän kalenterimerkintä -lomakkeen (Kuva 23) ulkonäkö suunniteltiin määrittelyn pohjalta.



Kuva 23. Työntekijän kalenterimerkintä

Työntekijän kalenterimerkinnän (Kuva 23) oli aluksi tarkoitus olla osa ohjelmaa ja loimmekin sille lomakkeen ohjelman osaksi, mutta asiaa palaverissa pohdittuamme totesimme sen melko hyödyttömäksi. Emme kuitenkaan poistaneet sitä kokonaan vaan, päätimme jättää sen suunnittelun myöhempään ajankohtaan.

## 9.9 Yleistä

Missään ohjelman poistamistoiminnossa ei tietoja poisteta tietokannasta, vaan ne merkitään tilaan, jossa ohjelma ei näytä niitä. Esimerkiksi, jos poistaa ohjelmalla työntekijän, niin työntekijän tiedot jäävät vielä tietokantaan, mutta ohjelma ei tule näyttämään niitä enää.

Jos tietokantayhteyttä ei saada auki, ohjelma ilmoittaa siitä käyttäjälle. Yhteyden aukioloa seurataan kaikissa tietokantaa käyttävissä toiminnoissa, eli lähes tulkoon jokaisessa toiminnossa. Yhteyden ollessa kiinni suurin osa toiminnoista ei toimi. Yhteyden uudelleen avatakseen tulee käyttäjän sulkea lomake ja avata se uudelleen Valikko-näkymästä.

## 10 Johtopäätökset

Tavoitteena oli suunnitella ja toteuttaa A-Katsastukselle työvuorojen hallintaohjelma. Tavoite saavutettiin.

Projektin aikana opimme paljon C#-kielen käyttämisestä, etenkin while-, foreach- ja for-looppien käyttäminen tuli erittäin tutuksi, sillä niitä täytyi käyttää erittäin paljon. Opimme paljon myös ohjelman kehittämisestä .NET-ympäristössä. Käytimme paljon Visual Studion valmiita luokkakirjastoja, joista löytyy valmiita luokkia ja metodeja eri komponenteille, jotka helpottivat työmääräämme paljon. Valmiiden luokkien ja metodien käyttäminen ei toisaalta ole paras vaihtoehto oppimisen kannalta, mutta niiden käyttäminen säästää erittäin paljon aikaa. SQL-kyselyjen tekemisessä harjaannuimme myös, koska teimme kaikki kyselyt manuaalisesti itse.

Ohjelman testaaminen tuli väkisinkin tutuksi tekemiämme virheitä etsiessä ja ohjelman toimintaa ja järkevyyttä pohtiessamme. Tutustuimme myös moniin erilaisiin testaustapoihin, joista kuitenkin itse työtä tehdessämme käytimme vain osaa, kuten regressiotestausta ja toiminnallista testausta.

Täysin uusi oppimamme asia oli transaktio (transaction), jota käytimme tietokannan tallentamisen yhteydessä. Toteutimme transaktiolla peruuta-toiminnon, johon se soveltui hyvin. Transaktiolla avulla tallensimme tiedot tietokantaan, jonka jälkeen transaktio voidaan vielä perua, jolloin tietokantaan ei tehdäkään muutoksia. Peruuta-toimintomme osaa kuitenkin peruuttaa vain yhden askeleen taaksepäin.

Ohjelmiston kehittämisestä opimme sen, että suunnittelu on erittäin tärkeää ja se kannattaa tehdä hyvin. Tämän opimme osittain erehdyksen kautta, sillä aluk-

si tekemämme ohjelmoinnit on tehty hieman kömpelömmiin kuin lopussa, ja jälkeinpäin katsottuna useita asioita olisi voinut tehdä siistimmin ja kätevämmiin.

Pariohjelmointi onnistui hyvin ja saimme toimintoja valmiiksi kohtuullisen nopeasti. Asiakkaan kanssa tiivis yhteydenpito toimi erittäin hyvin ja edes auttoi projektin valmistumista. Ohjelman testaaminen testauskurssilla oli erittäin hyödyllistä ja sen avulla ohjelmasta löytyi useita virheitä, jotka sitten korjattiin ja tämä opetti myös meitä ohjelmoijina.

Suunnittelu meni hieman heikosti, koska sitä tehtiin aivan liian vähän. Suunnittelun tasoa ja määrää olisi ollut helppo nostaa opiskelemalla enemmän aiheesta ja tiedostamalla sen tärkeyden.

Projektin dokumentointia tehtiin suhteellisen vähän ja projektin jatkoa ajatellen, sitä olisi kannattanut tehdä huomattavasti enemmän. Projektin loppuvaiheessa alkuperäisen yhteyshenkilömme menetys johti siihen, että ohjelman käyttöönotto ja testaaminen asiakkaalla pysähtyi.

Tietokantakyselyt teimme manuaalisesti SQL-kielellä, joka ei välttämättä ollut paras ratkaisu. Siihen olisi mahdollisesti kannattanut käyttää jotain valmista ORM-kehystä, kuten Entity Framework.

Jatkotoimenpiteinä ovat ohjelman testaaminen asiakkaan toimesta ja tämän jälkeen mahdollisten muutosten tekeminen ohjelmaan sekä ohjelman käyttöönotto.

## Kuvat

- Kuva 1. Valtioneuvoston asetus liikenteessä käytettävien ajoneuvojen liikenne-  
kelpoisuuden valvonnasta, s.10
- Kuva 2. Ohjelmistotuotannon osa-alueet, s.12
- Kuva 3. Vesiputousmalli, s.14
- Kuva 4. Protoilu, s.15
- Kuva 5. RUP, s.16
- Kuva 6. Scrum, s.20
- Kuva 7. Extreme programming, s.22
- Kuva 8. V-malli, s.28
- Kuva 9. 4+1 Kruchten, s.32
- Kuva 10. .NET arkkitehtuuri, s.38
- Kuva 11. SQL-elementit, s.43
- Kuva 12. IntelliSensen toiminta, s.46
- Kuva 13. Gantt-kaavio projektin vaiheista, s.48
- Kuva 14. Tietokantarakenne, s.51
- Kuva 15. Sisäänkirjautuminen, s.52
- Kuva 16. Valikko, s.53
- Kuva 17. Työvuorojen hallinta, s.54
- Kuva 18. Linjalista, s.55
- Kuva 19. Työntekijöiden hallinta, s.56
- Kuva 20. Merkintätyyppien hallinta, s.57
- Kuva 21. Merkintätyyppien vaatimukset, s.58
- Kuva 22. Toimipisteiden hallinta, s.59
- Kuva 23. Työntekijän kalenterimerkintä, s.60

## Lähteet

A-Katsastus Oy 2013

<http://a-katsastus.com/A-Katsastus-Group/Sivut/Konsernilyhyesti.aspx>. Luettu 20.11.2013.

Abrahamsson, P, Salo, O, Ronkainen, J & Warsta, J. 2002. Agile software development methods: review and analysis. VTT:n julkaisu 478. <http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf>.

Ajonaikaisen arkkitehtuurin kuvaaminen. 2009. Olioperustainen analyysi ja suunnittelu. Saimaan ammattikorkeakoulu.

Apache Friends. XAMPP

<http://www.apachefriends.org/en/xampp.html>. Luettu 29.11.2013.

C#-Station. C#

<http://csharp-station.com/>. Luettu 21.11.2013.

Don Wells. XP

<http://www.extremeprogramming.org>. Luettu 3.12.2013.

Edita Publishing Oy. Valtioneuvoston asetus liikenteessä käytettävien ajoneuvojen liikennekelpoisuuden valvonnasta 19.12.2002/1245 -kuva

<http://www.finlex.fi/fi/laki/ajantasa/2002/20021245>. Luettu 22.11.2013.

Haikala, I & Märijärvi, J. 2004. Ohjelmistotuotanto. Helsinki: Talentum Media Oy.

Haikala, I & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Helsinki: Talentum Media Oy.

IBM. RUP

<http://www-01.ibm.com/software/rational/rup>. Luettu 27.11.2013.

Kautto, T. 1996. Jyväskylän yliopisto. Ohjelmistotestaus.

<http://www.mit.jyu.fi/opiskelu/seminaarit/bak/testaus/>. Luettu 28.11.2013.

Kettunen, V. 2009. Ohjelmistotestaus ja ketterät menetelmät. Lappeenrannan teknillinen yliopisto.

Koskimies, K & Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Helsinki: Talentum Oyj

Microsoft. .NET

<http://www.microsoft.com/net>. Luettu 18.11.2013.

Microsoft. .NET-kuva

<http://msdn.microsoft.com/en-us/library/ms973842.aspx>



Microsoft. Access

<http://office.microsoft.com/en-us/access-help/database-basics-HA010064450.aspx>. Luettu 19.11.2013.

Microsoft. C#

<http://msdn.microsoft.com/en-us/vstudio/hh341490>. Luettu 21.11.2013.

Microsoft. Visual Studio

[http://msdn.microsoft.com/en-us/library/vstudio/dd831853\(v=vs.120\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/dd831853(v=vs.120).aspx). Luettu 2.12.2013.

Moghadampour, G. 2011. C# Windows- ja tietokantaohjelmointi. Helsinki: WSOYpro OY.

Nyström, T. 2005. Arkkitehtuurit ja koodin laatu. Sytyke-lehti 3/2005, s.8-12. <http://www.pcuf.fi/sytyke/lehti/kirj/st20053/ST053-08A.pdf>. Luettu 8.1.2014.

Oracle. MySQL

<http://www.mysql.com/>. Luettu 26.11.2013.

Oulun seudun ammattiopisto. Vesiputousmalli-kuva

[http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien\\_kaytto\\_ja\\_kehittamien/johdatus\\_tietojarjestelmiin/kehittamistyon\\_vaiheet\\_ja\\_elikaarimallit/kehittamistyon\\_vaiheet\\_ja\\_elinkaarimallit\\_asia.htm](http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittamien/johdatus_tietojarjestelmiin/kehittamistyon_vaiheet_ja_elikaarimallit/kehittamistyon_vaiheet_ja_elinkaarimallit_asia.htm)

Scrum

<https://www.scrum.org>. Luettu: 28.11.2013.

Työaikalaki 9.8.1996/605

W3Schools. SQL

<http://www.w3schools.com/sql/>. Luettu 29.11.2013.

Wikipedia. Kruchten 4+1-kuva

[http://en.wikipedia.org/wiki/File:4%2B1\\_Architectural\\_View\\_Model.jpg](http://en.wikipedia.org/wiki/File:4%2B1_Architectural_View_Model.jpg)

Wikipedia. RUP-kuva

<http://fi.wikipedia.org/wiki/Tiedosto:IteratiivisenMallinVaiheet.JPG>

Wikipedia. SQL-elementit

[http://en.wikipedia.org/wiki/File:SQL\\_ANATOMY\\_wiki.svg](http://en.wikipedia.org/wiki/File:SQL_ANATOMY_wiki.svg)